Listat ja aggregaatio NoSQL-dokumenttivarastossa – tapaus Taitorakennerekisteri

Tuuli Kähkönen

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Pro gradu -tutkielma
Ohjaaja: Kati Iltanen ja Marko Junkkari
Kesäkuu 2017

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Tuuli Kähkönen: Listat ja aggregaatio NoSQL-dokumenttivarastossa – tapaus Taitorakennerekisteri
Pro gradu -tutkielma, 45 sivua
Kesäkuu 2017

Tässä tutkimuksessa tarkastellaan NoSQL-tietokantojen ja niitä muistuttavien hakukoneiden ilmaisuvoimaa erityisesti alidokumentteihin kohdistuvissa hauissa. NoSQL-tietokannoilla tarkoitetaan relaatiomallista poikkeavia tietokantoja. Dokumentilla tarkoitetaan avain-arvo-pareista koostuvaa tietorakennetta. Dokumentilla voi olla myös dokumenttityyppisiä arvoja tai listatyyppisiä arvoja, jotka sisältävät dokumentteja, joita kutsutaan tässä yhteydessä alidokumenteiksi. Tutkimuksen menetelminä ovat tapaustutkimus ja konstruktiivinen tutkimus. Tarkasteltavana tapauksena tutkimuksessa on Liikenneviraston Taitorakennerekisteri-tietojärjestelmän hakutoiminnallisuus, joka pohjautuu Elasticsearch-hakukoneeseen.

Tutkimuksessa halutaan vastata siihen, miten yleisesti NoSQL-tietokannoilla ja erityisesti Elasticsearchilla voi toteuttaa alidokumentteja koskevia hakuja ja toisaalta miten ilmaisuvoimainen haku toteutuu tarkastellun tapauksen tietojärjestelmässä. Tutkimuksessa kartoitetaan, miten kirjallisuudessa on käsitelty NoSQL-tietokantoja ja niiden ilmaisuvoimaa alidokumentteihin kohdistuvissa ja aggregaatiota hyödyntävissä hauissa. Tutkimuksen konstruktiivisessa osuudessa toteutettiin hakutoiminnallisuuden laajennus Taitorakennerekisteri-järjestelmään, jolla on tarkoitus parantaa haun ilmaisuvoimaa.

Avainsanat ja -sanonnat: tiedonhaku, aggregoinnit, tietokannat, NoSQL, hakukoneet.

Sisällys

1. Johdanto 1
2. Tutkimuskysymykset ja tutkimusmenetelmät 3
   2.1. Tutkimuskysymykset 3
   2.2. Tutkimusmetodit 3
      2.2.1. Tapaustutkimus 3
      2.2.2. Konstruktiivinen tutkimus 5
3. NoSQL-tietokannat ja hakukoneet 7
   3.1. Relaatiotietokannat 7
   3.2. NoSQL-tietokantojen piirteitä ja luokittelua 7
   3.3. Aggregointi ja liitosehdot NoSQL-tietokannoissa 12
   3.4. Elasticsearch 14
4. Taitorakennerekisteri 16
   4.1. Taitorakennerekisterin tavoitteet ja tietomalli 16
   4.2. Haku Taitorakennerekisterissä 18
      4.2.1. Elasticsearch Taitorakennerekisterissä 21
   4.3. Taitorakennerekisterin haun rajoitteet ja tarve lisätä ilmaisuvoimaa 27
5. Elasticsearch-haun laajentaminen Taitorakennerekisterissä 29
   5.1. Haun laajentamistarpeet 29
   5.2. Vertailua kyselyihin SQL-kielellä 30
   5.3. Ratkaisun tekninen toteutus 32
6. Toteutetun ratkaisun arviointia 37
7. Yhteenveto 40
Viiteluettelo 41

1. Johdanto

Toimiva tiedonhaku on olennainen osa useita tietojärjestelmiä. Etenkin nykyään, kun dataa tuotetaan ja käsitellään entistä enemmän, tehokkaan parametrisen tiedonhakutoiminnallisuuden merkitys korostuu yhä useammissa tietojärjestelmissä. Parametrisella tiedonhaulla tarkoitetaan hakua, jossa (kontrastina tekstihaulle) käyttäjä kohdistaa haun tiettyihin dokumentin attribuutteihin [Tunkelang, 2009]. Parametrista hakua käytetään laajasti erilaisissa tietojärjestelmissä. Esimerkiksi verkkokaupoissa parametrinen haku on tärkeä ominaisuus, jotta ostaja löytää helpommin haluamansa tuotteen. Myös kokotekstihakuun perustuvat hakukoneet hyödyntävät parametrista hakua tarkennetun haun toiminnoissaan.

Tietojärjestelmien kehittäjille on markkinoilla runsaasti erilaisia valmiita hakukoneita, jotka voi upottaa osaksi omaa sovellusta ja jotka voi säätää toimimaan oman sovelluksen määrittelyjen vaatimalla tavalla. Valmiiden hakukoneiden käyttö nopeuttaa huomattavasti tietojärjestelmän kehitystyötä. Valmiit hakukoneet ovat usein tehokkaita käytössä ja sopivat hyvin erityisesti hakuun tekstidokumenteista. Usein ne tukevat myös rakenteista dataa ja monet niistä muistuttavat toiminnallisuudeltaan NoSQL (*ei vain SQL)* -tietokantoja. NoSQL-tietokannat luokitellaan niiden tietomallin mukaan seuraavasti: avain-arvo-varastot, dokumenttivarastot, sarakeperhevarastot ja graafitietokannat [Hecht and Jablonski, 2011].

Tässä tutkimuksessa tarkastellaan dokumenttivarastoa muistuttavaa hakukonetta, Elasticsearchia. Dokumenttivarastossa tieto tallennetaan hierarkkisesti JSON-muotoisina dokumentteina [Hecht and Jablonski, 2011]. Dokumentti sisältää avain-arvo-pareja. Arvot voivat dokumenttivarastossa sisältää myös alidokumentteja tai alidokumenttilistoja. Hierarkialla ilmennetään dokumenttien välisiä suhteita. Elasticsearchissa haku alidokumenttilistoista on mahdollista, mutta rajoittuu ”dokumentit, joiden yksi tai useampi alidokumentti täyttää ehdon” -tyyppisiin hakuihin. Monipuolisempien ehtojen alidokumenteille antaminen tai alidokumenteista aggregoidun tiedon perusteella suodattaminen ei ole mahdollista Elasticsearchin kyselykielen puitteissa.

Relaatiotietokannoissa vastaavia alikäsitteisiin kohdistuvia hakuja tehdään liitosoperaatioiden avulla [Kroenke and Auer, 2016]. Alikäsitteen suhde pääkäsitteeseen voi olla joko isA-suhde tai kokonaisuus-osa-suhde. Relaatiotietokannoissa käsitteiden väliset suhteet ilmaistaan vierasavaimilla. IsA-suhteessa, joka on 1:1 -suhde, alikäsitteen tauluun laitetaan vierasavain, joka osoittaa pääkäsitteen tauluun. Kokonaisuus-osa-suhde on 1:M -suhde, jossa siinäkin alikäsitteen taulussa vierasavain osoittaa pääkäsitteen tauluun.

Tämän tutkimuksen aiheena on Elasticsearch-haun laajentaminen tarkempiin alidokumentteja koskeviin hakuihin oikeassa käyttötapauksessa. Tarkasteltavana tapauksena

on Liikenneviraston Taitorakennerekisteri-tietojärjestelmä ja sen laaja hakutoiminto. Elasticsearchin kyselykieli on puutteellinen alidokumentteihin kohdistuvien kyselyiden kohdalla ja siksi kaikkia järjestelmän sovellusalueella tarvittavia kyselyitä ei pysty sillä suorittamaan. Tutkimuksessa toteutettiin järjestelmään hakutoiminnallisuuden laajennus, jonka avulla halutun tyyppisiä kyselyitä pystytään suorittamaan järjestelmän Elasticsearch-pohjaisella haulla.

Kirjallisuudessa on jonkin verran käsitelty hakua alikäsitteistä ja erilaisten SQL-kielelle tyypillisten kyselyiden vastineita NoSQL-tietokannoissa [Rith *et al.*, 2014]. Kirjallisuudessa on myös esitetty muutamia ratkaisuja alidokumentteihin kohdistuvien hakujen toteuttamiseen NoSQL-tietokannoissa [Pasqualin *et al.*, 2016; Atzeni *et al*., 2012]. Tässä tutkimuksessa päädyttiin kuitenkin Elasticsearchin erityispiirteiden vuoksi näistä lähteistä poikkeavaan ratkaisuun, jossa hyödynnetään Elasticsearchin skriptausominaisuutta.

Tutkimuksen toisessa luvussa esitellään tutkimuksen tutkimuskysymykset ja tutkimusmetodit tapaustutkimus ja konstruktiivinen tutkimus. Kolmannessa luvussa kuvaillaan ja määritellään NoSQL-tietokantoja yleisesti, esitellään niiden tärkeimpiä piirteitä ja miten NoSQL-tietokantoja on kirjallisuudessa kategorisoitu. Luvussa neljä esitellään tutkimuksen kohteena oleva tapaus: Taitorakennerekisteri-järjestelmä ja sen hakutoiminnallisuus. Luvussa kuvaillaan Elasticsearch-hakukoneen ominaisuuksista johtunut haun tiettyjä kyselyjä koskeva ilmaisuvoiman puute ja tämän pohjalta tarpeet toiminnallisuuden laajentamiseen. Hakutoiminnallisuuden laajennuksen periaate ja tekninen toteutus esitellään luvussa viisi. Luvussa kuusi arvioidaan toteutettua ratkaisua. Luku seitsemän on yhteenveto.

2. Tutkimuskysymykset ja tutkimusmenetelmät

Tässä luvussa kuvataan tutkimuksen tutkimuskysymykset. Luvussa esitellään myös tutkimuksessa käytetyt tutkimusmetodit, joiksi valittiin tapaustutkimus ja konstruktiivinen tutkimus.

2.1. Tutkimuskysymykset

Tutkimuksen aiheen hahmottelun pohjana oli Liikenneviraston Taitorakennerekisteri-tietojärjestelmän (luku 4) Elasticsearch-hakukoneella (kohta 3.4) toteutettu hakutoiminnallisuus ja tarve lisätä sen ilmaisuvoimaa. Haun ilmaisuvoiman lisäämisen tavoitteena oli mahdollistaa monipuolisempien alidokumentteihin kohdistuvien kyselyjen tekeminen. Tavoitteena oli tuottaa monipuolisemman haun mahdollistava ratkaisu Taitorakennerekisterin yhteyteen ja tutkia, miten NoSQL-tietokantoja (luku 3) hyödyntävissä järjestelmissä voidaan mahdollistaa alidokumentteihin kohdistuvia hakuja.
Tutkimuksen lähtökohdista nousee esille seuraavat kaksi tutkimuskysymystä:

1. Miten yleisesti NoSQL-tietokannoilla ja erityisesti Elasticsearchilla voi toteuttaa alidokumentteja koskevia hakuja?
2. Miten alidokumentteja koskeva Elasticsearch-haku toteutetaan Taitorakennerekisterissä?

Ensimmäiseen tutkimuskysymykseen vastaamiseksi kuvaillaan erilaisia kirjallisuudessa mainittuja tapoja ratkaista alidokumentteihin kohdistuvia hakuja. Toiseen kysymykseen vastaamiseksi esitellään ratkaisun toteutus ja pohditaan myös muita mahdollisia ratkaisuvaihtoehtoja.

2.2. Tutkimusmetodit

Tutkimusmetodeiksi valittiin tapaustutkimus ja konstruktiivinen tutkimus. Tapaustutkimus ja konstruktiivinen tutkimus valittiin, koska haluttiin konkreettisesti selvittää, miten tiettyä hakujärjestelmää voidaan monipuolistaa alidokumentteihin kohdistuvien hakujen osalta. Konstruktiivisen tutkimuksen apuna käytettiin menetelmänä myös kirjallisuuskatsausta.

2.2.1. Tapaustutkimus

Tapaustutkimuksessa eli case-tutkimuksessa käsitellään joko yhtä tai useampaa tapausta. Kerättävä tieto voi olla sekä kvantitatiivista että kvalitatiivista. Tapaustutkimuksessa voidaan kuvailla tiettyä tapausta tai tapauksia. Useampaa tapausta käsittelevässä tapaustutkimuksessa joko vertaillaan useaa samankaltaista tapausta tai pyritään löytämään selitys

eroille tapauksissa [Yin, 2003]. Myös teorian testaaminen tai uuden teorian luominen on mahdollista tapaustutkimuksessa [Järvinen ja Järvinen, 2001]. Tapaustutkimus vastaa ”miten?” ja ”miksi?” -tyyppisiin kysymyksiin [Yin, 2003].

Yinin [2003] mukaan tapaustutkimus on hyvä menetelmävalinta, jos tutkimuksen kohteen käyttäytymiseen ei pysty vaikuttamaan. Tapaustutkimus kannattaa valita myös silloin, jos kontekstin ajatellaan olevan tutkittavan ilmiön kannalta erityisen merkityksellinen ja sen takia se halutaan sisällyttää tutkimukseen. Tapaustutkimusta kannattaa käyttää menetelmänä myös, jos rajanveto tutkittavan ilmiön ja sen kontekstin välillä ei ole selvää.

Yinin [2003] mukaan tapaustutkimus voidaan kategorisoida selittävään (*explanatory*), etsivään (*exploratory*) ja kuvailevaan (*descriptive*) tapaustutkimukseen. Selittävässä tapaustutkimuksessa pyritään löytämään selitys oletetulle kausaaliselle suhteelle. Menetelmää voidaan käyttää, jos esimerkiksi kyselytutkimus tai kokeellinen tutkimus eivät olisi mahdollisia tai käytännöllisiä käyttää tutkittavan ilmiön monimutkaisuuden vuoksi. Etsivässä tapaustutkimuksessa tutkittavalla interventiolla ei ole yksiselitteistä lopputulosta. Kuvailevassa tapaustutkimuksessa kuvaillaan ilmiö ja sen konteksti tosielämässä.

Stake [1995] puolestaan luokittelee tapaustutkimuksen itseisarvoiseen (*intrinsic*) ja välineelliseen (*instrumental*) tapaustutkimukseen. Itseisarvoisessa tutkimuksessa tapausta tutkitaan, koska se on itsessään erikoinen tai mielenkiintoinen. Tapauksen perusteella ei yritetä vahvistaa tai muodostaa laajempaa teoriaa. Välineellisessä tapaustutkimuksessa tapaus itsessään ei ole mielenkiinnon kohteena, vaan sen avulla pyritään ymmärtämään laajempaa ilmiötä tai luomaan teoriaa. Välineellisessä tapaustutkimuksessa tutkimuksen kohteeksi saatetaan valita ilmiölle erityisen tyypillinen tai edustava tapaus, mutta se ei ole aina välttämätöntä.

Tapaustutkimuksessa voidaan käyttää laajasti eri tietolähteitä. Mahdollisia tapaustutkimuksessa käytettäviä tietolähteitä ovat muun muassa dokumentaatio, arkistot, haastattelu, artefaktit, tarkkailu ja osallistuva tarkkailu (*participant-observation*) [Baxter and Jack, 2008]. Monipuolista tietolähteiden käyttöä pidetään yhtenä tapaustutkimuksen vahvuuksista.

Tapaustutkimusta on myös kritisoitu tutkimusmenetelmänä [Yin, 2003]. Kriitikoiden mukaan tapaustutkimuksessa ei ole tieteellistä kurinalaisuutta. Tapaustutkimusta on kritisoitu myös siitä, etteivät yhtä tapausta koskevat tulokset ole yleistettävissä. Tapaustutkimus vaatii myös paljon resursseja ja on myös tutkijalle vaativa menetelmä.

2.2.2. Konstruktiivinen tutkimus

Konstruktiivisessa tutkimuksessa rakennetaan uusia innovaatioita olemassa olevan tutkimustiedon pohjalta [Järvinen ja Järvinen, 2001]. Innovaatiolla tarkoitetaan ihmisen tai ihmiskollektiivin tavoittelemaa uudistusta, josta toivotaan olevan käyttäjälleen hyötyä.

Konstruktiivinen tutkimus on soveltavaa tutkimusta, jossa ei pyritä selittämään maailman ilmiöitä, kuten perustutkimuksessa, vaan pyritään pääsemään hyödylliseen lopputulokseen. Hyödyllisen lopputuloksen määrittely on luonteeltaan subjektiivista ja se riippuu tutkijan tai muiden päätöksentekijöiden arvoista. Tästä syystä konstruktiivinen tutkimus on aina normatiivista. Sen sijaan, että kuvailtaisiin, miten asiat ovat, konstruktiivisessa tutkimuksessa keskitytään siihen, miten asioiden pitäisi olla. Tutkija voi Järvisen ja Järvisen [2001] mukaan konstruktiivisessa tutkimuksessa olla erilaisissa rooleissa. Usein tutkija on itse toteuttamassa tutkimuksessa rakennettavaa innovaatiota. Joskus tutkija kuitenkin tekee tutkimusta hankkeen ulkopuolelta.

Konstruktiivisen tutkimuksen tulokset voivat olla joko käsitteistöjä, malleja, metodeja tai realisointeja [March and Smith, 1995]. Käsitteistöllä kuvataan sekä nykyhetkeä vahvuuksineen ja heikkouksineen sekä tavoiteltua lopputulosta [Järvinen ja Järvinen, 2001]. Mallilla ilmaistaan käsitteiden väliset suhteet. Metodi on jäsennetty toimintasuunnitelma, jolla tavoitellaan siirtymää alkutilanteesta toivottuun lopputulokseen. Realisointi on toteutettu artefakti ympäristössään.

Järvisen ja Järvisen [2001] mukaan on kaksi vaihtoehtoista tapaa toteuttaa innovaatio. Ensimmäisessä tavassa ensin spesifioidaan tavoiteltu lopputulos, minkä jälkeen seuraa implementointiprosessi. Jälkimmäisessä tavassa spesifiointi- ja implementointiprosessit ovat rinnakkaiset. Tavoiteltua lopputulosta pyritään lähenemään esimerkiksi jatkuvasti kehittyvien ja tarkentuvien prototyyppien avulla.

Olennainen osa konstruktiivista tutkimusta on tuotetun innovaation arviointi. Mikäli aikaisempi vastaava innovaatio on jo olemassa, on tuloksia verrattava tähän aikaisempaan vastaavaan käsitteistöön, malliin, metodiin tai realisaatioon ja erityisesti arvioitava, onko uuden innovaation hyödynnettävyys suurempi kuin vanhan [March and Smith, 1995]. Järvinen ja Järvinen [2001] lisäävät tähän, että uutta innovaatiota on aina verrattava parhaaseen mahdolliseen vanhaan innovaatioon. Jos kyseessä taas on täysin uusi innovaatio, Marchin ja Smithin mukaan se on saavutus jo itsessään. Järvinen ja Järvinen kuitenkin täydentävät, että myös aidosti uuden innovaation hyödynnettävyyttä on yritettävä arvioida.

Eri tyyppisiä konstruktiivisen tutkimuksen tuloksia pitää arvioida eri kriteereillä. Marchin ja Smithin [1995] suosittelemat mittarit käsitteistölle konstruktiivisen tutkimuksen tuloksena ovat täydellisyys, yksinkertaisuus, eleganssi, ymmärrettävyys ja helppo-

käyttöisyys. Järvinen ja Järvinen [2001] pitävät näistä ainoastaan helppokäyttöisyyttä järkevänä mittarina, koska muilla kriteereillä ei ole vaikutusta tutkimuksen tuloksen hyödynnettävyyteen, joka on konstruktiivisessa tutkimuksessa olennaista. Järvinen ja Järvinen korostavatkin hyödynnettävyyttä käsitteistön keskeisenä arviointikriteerinä.

Mallin arviointikriteereiksi March ja Smith [1995] antavat mallin ja todellisuuden vastaavuuden, täydellisyyden, yksityiskohtaisuuden, lujuuden ja sisäisen johdonmukaisuuden. Järvinen ja Järvinen [2001] eivät pidä mallin ja todellisuuden vastaavuutta järkevänä kriteerinä, koska jos malli on konstruktiivisen tutkimuksen tulos, sitä ei ole vielä tässä vaiheessa toteutettu. Heidän mukaansa myös täydellisyys ja yksityiskohtaisuus ovat ongelmallisia kriteerejä, koska mallin rakentaminen perustuu joka tapauksessa abstrahointiin. Malli ei koskaan voi olla täysin täydellinen ja liiallinen yksityiskohtiin pyrkiminen tekee mallista helposti liian laajan.

Metodin arviointikriteereitä ovat operationaalisuus, tehokkuus, yleisyys ja helppokäyttöisyys [March and Smith, 1995]. Järvinen ja Järvinen [2001] lisäävät näihin vielä metodin sovellusmahdollisuuksien arvioinnin: millaisella sovellusalueella metodia voi käyttää, sekä metodin käyttöön tarvittavat tekniset, inhimilliset tai tiedolliset resurssit.

Realisaation arviointikriteereiksi March ja Smith [1995] antavat artefaktin tehokkuuden ja vaikuttavuuden sekä sen vaikutukset ympäristöön ja käyttäjiin. Järvinen ja Järvinen [2001] nostavat esille myös artefaktin käyttöönoton (esimerkiksi sosiaaliset) vaikutukset laajemminkin kuin sen lähimpään ympäristöön ja käyttäjiin. He antavat uudeksi arviointikriteeriksi myös huollon vaikutukset, esimerkiksi sen kuinka paljon kehitetty tietojärjestelmä tulee elinkaarensa aikana tarvitsemaan huoltoa ja kehottavat arvioimaan myös sen käytöstä poistamisen vaikutuksia.

3. NoSQL-tietokannat ja hakukoneet

Tässä luvussa esitetään NoSQL-tietokantojen yleisiä piirteitä ja miten NoSQL-tietokantoja luokitellaan niiden ominaisuuksien perusteella. Erityisesti huomioidaan tutkimuksen kannalta olennainen hakukone Elasticsearch ja miten se sijoittuu NoSQL-tietokantojen kentällä. Myös relaatiotietokannat ja niihin liittyvät käsitteet esitellään luvun alussa. Luvussa pohditaan myös, miten aggregointia ja alidokumentteihin liittyviä hakuja NoSQL-tietokannoissa on aiemmin tutkittu ja minkälaisia ratkaisuja on tehty helpottamaan erityisesti aggregointi- ja alidokumenttihakuja.

3.1. Relaatiotietokannat

Relaatiotietokannat ovat yleisimmin käytössä olevia tietokantoja. Relaatiotietokantoja alettiin kehittää 1970-luvulla. Alun perin relaatiomallin kehitti Codd [1970].

Relaatiotietokannan looginen rakenne ilmaistaan *tietokantakaaviossa* eli *skeemassa.* Skeema sisältää tietokannan *taulut*, *sarakkeet*, *pääavaimet*, *vierasavaimet* ja muun tietokannan loogisen rakenteen. Yksi taulu vastaa yleensä yhtä reaalimaailman entiteettityyppiä tai suhdetyyppiä entiteettien välillä. Jokainen taulun *rivi* taas vastaa yhtä reaalimaailman entiteettiä tai suhdetta. Taulun sarake on kaikille taulun riveille yhteinen attribuutti, jolle on määritelty myös tietotyyppi. Taulun pääavain on rivin yksilöivä sarake tai sarakkeiden yhdistelmä. Relaatiotietokannassa kaikkien taulun rivien on oltava keskenään erilaisia.

Suhteet taulujen välillä ilmaistaan vierasavaimilla. Vierasavain viittaa toisen taulun pääavaimeen. Useista tauluista voi koota tietoa tulostauluun yhdistämällä rivejä *liitos-operaatioilla*. Liitokselle annetaan ehdot, joiden mukaan rivien pitää vastata toisiaan, jotta tulokseen saadaan mukaan vain mielekkäät rivit. *Sisäliitoksessa* mukaan tulostauluun tulevat kummastakin taulusta vain rivit, jotka vastaavat liitosehtoja. *Ulkoliitoksessa* puolestaan mukaan otetaan joko toisesta tai kummastakin taulusta myös rivit, jotka eivät vastaa liitosehtoa. [Kroenke and Auer, 2016]

Relaatiotietokannassa käsitettä ja alikäsitettä tai listatyyppistä arvoa vastaava tietorakenne ilmaistaan siten, että alikäsitteen tauluun laitetaan vierasavainsarake, joka viittaa pääkäsitteen tauluun. Näin yksi pääkäsitteen taulun rivi on suhteessa nollaan tai useampaan alikäsitteen riviin.

3.2. NoSQL-tietokantojen piirteitä ja luokittelua

Cattellin [2011] mukaan termin NoSQL ajatellaan yleensä tarkoittavan kirjaimellisesti *ei vain SQL* (Not Only SQL) tai *ei relationaalinen.* Cattellin mukaan yksi NoSQL-tietokantojen yleisimpiä piirteitä on niiden skaalautuvuus useiden palvelinten välille. Kun uusia

palvelimia lisätään, järjestelmän tehokkuus lisääntyy. Tällä niin kutsutulla *horisontaalisella skaalautuvuudella* saadaan tehokkuuden lisäksi parannettua järjestelmän saatavuutta, mikä on erittäin tärkeää sovellusalueilla, joilla NoSQL-tietokantoja käytetään [Cattell, 2011]. Horisontaalisesti skaalautuvassa järjestelmässä tietokantajärjestelmän sisältämä data voidaan monistaa tai osittaa eri palvelimille. Olennaista on, että tietokantajärjestelmän dataa sisältävät solmut eivät jaa keskenään muistia, toisin kuin *vertikaalisessa skaalautuvuudessa,* jossa tehokkuutta lisätään lisäämällä yhden palvelimen suorituskykyä. Vertikaalinen skaalautuminen on ominaista perinteisille relaatiotietokantajärjestelmille.

Horisontaalisen skaalautuvuuden saavuttamiseksi NoSQL-tietokantajärjestelmissä käytetään usein joko *horisontaalista* tai *vertikaalista osittamista*. [Cattell, 2011]. Horisontaalisessa osittamisessa tietokannan sisältämät rivit on jaettu eri palvelimille jonkin avaimen mukaan. Esimerkiksi käyttäjätiedot sisältävä taulu voitaisiin järjestää käyttäjän sukunimen mukaan ja sitten jakaa eri palvelinten kesken. Vertikaalisessa osittamisessa taas saman taulun eri sarakkeet jaetaan eri palvelimille. Horisontaalisen osittamisen hyötynä on yksittäisen rivin tietojen nopea haku, vertikaalisen puolestaan erilaisten aggregointitietojen nopea hakeminen. Tietoa voidaan jakaa eri palvelimille myös hajautusfunktion avulla. [Pokorný, 2015] Osittaminen tapahtuu NoSQL-tietokannoissa automaattisesti; järjestelmä osaa itse päätellä datan todennäköisesti tehokkaimman osituksen, eikä tietokannan käyttäjän tarvitse yleensä huolehtia siitä.

NoSQL-tietokannoissa tehokkuutta on parannettu eheyden kustannuksella [Cattell, 2011]. Relaatiotietokantojen transaktio-ominaisuuksia kuvaillaan ACID-lyhenteellä, joka tulee sanoista *atomisuus* (atomicity), *eheys* (consistency), *eristyneisyys* (isolation) ja *pysyvyys* (durability) [Haerder and Reuter, 1983]. ACID-periaate pitää huolta siitä, että transaktioihin kuuluvat toiminnot suoritetaan kokonaisuudessaan ja että tietokannan tila säilyy siten johdonmukaisena. ACID-periaatteen sijasta NoSQL-tietokannat toteuttavat usein BASE-periaatetta [Pokorný, 2015]. BASE on lyhenne, joka tulee sanoista *periaatteessa käytettävissä* (basically available), *ei aina oikeellinen* (soft state) ja *lopulta eheä* (eventually consistent). BASE-periaatteella toimivassa tietokannassa tietokannan tila saattaa hetkellisesti olla virheellinen, mutta lopulta tila korjaantuu eheäksi.

Hecht ja Jablonski [2011] luokittelevat NoSQL-tietokannat tietomallin perusteella neljään luokkaan. Tietomallin perusteella luokittelu on perusteltua, koska NoSQL-tietokannat eroavat relaatiotietokannoista juuri tietomalliltaan. Hechtin ja Jablonskin käyttämät luokat ovat avain-arvo-varastot, dokumenttivarastot, sarakeperhevarastot ja graafitietokannat.

Avain-arvo-varastoissa on hyvin yksinkertainen tietomalli, joka muistuttaa assosiaatiotaulua tai sanakirjaa. Yksi avain liitetään yhteen arvoon, jolloin arvon hausta

avaimella tulee hyvin nopeaa. Avain-arvo-varastoissa ei ole skeemaa, ja avain-arvo-varastotietokantoihin voi dynaamisesti lisätä uusia, tyypiltään vaihtelevia, avain-arvopareja [Hecht and Jablonski, 2011]. Avain-arvo-varastot ovat kuitenkin usein tehottomia tilanteissa, joissa arvot sisältävät esimerkiksi assosiaatiotauluja tai listoja, ja on tarve lukea tai päivittää näitä arvojen sisältämiä syvempiä rakenteita [Pokorný, 2015]. Avain-arvo-varastoja ovat muun muassa Amazonin SimpleDB [Amazon Web Services, 2017], Memcached [Memcached, 2017], Redis [Redis, 2017], Project Voldemort [Project Voldemort, 2017], Riak [Basho, 2017] ja Scalaris [Zuse Institute Berlin, 2017]. Hecht ja Jablonski [2011] tosin luokittelevat Riakin dokumenttivarastoksi avain-arvo-varaston sijaan, koska sen tietorakenteessa voi olla olioita, joilla on useita avaimia. Cattell [2011] taas luokittelee SimpleDB:n dokumenttivarastoksi.

Dokumenttivarastojen tietomallissa data tallennetaan joko JSON-muodossa tai läheisesti JSON-notaatiota muistuttavalla tavalla. JSON-notaatio kuvaa JavaScript-ohjelmointikielen oliota, ja sitä käytetään monissa web-sovelluksissa välittämään dataa palvelimen ja selaimen välillä [Bray, 2014]. JSON-dokumentin sisällä avaimet ovat uniikkeja. Dokumenttivarastossa arvot voivat sisältää listoja ja alidokumentteja, ja myös haku ja päivitys alidokumentteihin on mahdollista. Hechtin ja Jablonskin [2011] mukaan dokumenttivarastot ovat avainarvovarastojen tapaan skeemattomia ja uusia avaimia voi tallentaa dokumentteihin dynaamisesti. Dokumenttivarastoja ovat muun muassa MongoDB [MongoDB Inc, 2017], OrientDB [OrientDB Ltd, 2017] ja CouchDB [Apache Software Foundation, 2017a].

Sarakeperhevarastojen tietomallissa rivi voi sisältää mitä tahansa sarakkeita. Riveistä muodostetaan sarakeperheitä, jotka tallennetaan aina samaan tiedostoon, jolloin niiden hakeminen nopeutuu. Riviin voi lisätä dynaamisesti uusia sarakkeita, mutta sarakeperheet tulee määritellä etukäteen [Cattell, 2011]. Sarakeperhevarastojen etu relaatiomalliin nähden on se, että jokaisella rivillä on omat sarakkeensa, jolloin tietokanta ei sisällä tyhjäarvoja. Usein haku sarakeperhevarastotietokannoista tapahtuu SQL-kieltä muistuttavalla kyselykielellä, joka ei kuitenkaan tue liitosoperaatioita. [Hecht and Jablonski, 2011] Sarakeperhevarastojen tietomalli mahdollistaa sekä rivien että sarakkeiden osittamisen eri palvelimille, joten sarakeperhevarastot ovat hyödyllisimmillään silloin, kun käsitellään suurta määrää dataa [Cattell, 2011]. Ensimmäinen ja tunnetuin sarakeperhevarasto on Googlen BigTable [Chang *et al*., 2008]. HBase [Apache Software Foundation, 2017b] ja Hypertable [Hypertable Inc, 2017] ovat avoimen lähdekoodin toteutuksia BigTablen pohjalta. Cassandra [Apache Software Foundation, 2017c] on myös sarakeperhevarasto, mutta lisää BigTablen tietomalliin *supersarakkeen* käsitteen. Supersarake voi sisältää useita tavallisia sarakkeita [Hecht and Jablonski, 2011].

Graafitietokannat järjestävät datan solmuiksi ja kaariksi. Graafitietokannoissa käytetään usein skeemaa, jolla määritellään, minkä tyyppisiä kaaria, eli suhteita, solmujen, eli entiteettien välillä voi olla. Solmuihin ja kaariin voi usein myös liittää tietoa avain-arvo-pareilla. Graafitietokannat ovat tehokkaita käyttötapauksissa, joissa entiteettien väliset suhteet ovat tärkeitä, kuten esimerkiksi sosiaalisen median palveluissa, joissa käyttäjäprofiilien välillä on erilaisia suhteita. Graafissa suhteiden hakeminen on nopeaa, kun taas relaatiotietokannassa suhteet pitää hakea tehottomampien liitosoperaatioiden avulla [Hecht and Jablonski, 2011]. Graafitietokantoja ovat Neo4j [Neo Technology Inc, 2017] ja GraphDB [Ontotext, 2017].

Relaatiotieto-kannat	**Avain-arvo-varasto**	**Dokumentti-varasto**	**Sarakeperhe-varasto**	**Graafitieto-kanta**
Tietokantakaa-vio (skeema)	(joillakin) ku-vaus	(joillakin) ku-vaus	*ei vastaavuutta*	(joillakin) ku-vaus
Taulu	*ei vastaavuutta*	*ei vastaavuutta*	sarakeperhe	*ei vastaavuutta*
Rivi	tietue	dokumentti	avain-arvo-pari	solmu
Sarake	avain	avain	sarake	ominaisuus (property)
Näkymä	*ei vastaavuutta*	*ei vastaavuutta*	supersarake	*ei vastaavuutta*

Taulukko 1. Relaatiotietokantojen ja NoSQL-tietokantojen käsitteiden vastaavuuksia tietomallikohtaisesti.

Taulukossa 1 etsitään vastaavuuksia relaatiotietokantojen käsitteiden ja NoSQL-tietokantojen käsitteiden välille. Koska tietomallit eroavat suuresti toisistaan, vastaavuudet eivät ole täsmällisiä, vaan suuntaa antavia. Tietokantakaaviota vastaa avain-arvo-varastoissa, dokumenttivarastoissa ja graafitietokannoissa kuvaus, jossa määritellään sallitut avaimet tai ominaisuudet ja niiden tietotyypit. Sarakeperhevarastoissa ei ole tietokantakaaviota vastaavaa käsitettä tai toiminnallisuutta. Relaatiotietokantojen taulua vastaa sarakeperhevarastoissa sarakeperhe. Taulun riviä vastaavat käsitteet tietue avain-arvo-varastoissa, dokumentti dokumenttivarastoissa, avain-arvo-pari sarakeperhevarastoissa ja solmu graafitietokannoissa. Relaatiotietokantojen saraketta vastaa avain-arvo-varastoissa ja dokumenttivarastoissa avain, sarakeperhevarastoissa sarake ja graafitietokannoissa solmun ominaisuus. Relaatiotietokantojen näkymälle löytyy vastaavuus vain Cassandra-sarakeperhevaraston supersarakkeesta.

Ensimmäisenä modernina NoSQL-tietokantana pidetään Amazonin Dynamoa [DeCandia, 2007], joka otettiin käyttöön vuonna 2006. NoSQL-tietokantojen käyttö yleistyi

ja uusia toteutuksia ilmestyi paljon vuoden 2010 tienoilla [Mohan, 2013]. Mohan huomauttaa, että on olemassa myös paljon vanhempia ohjelmistoja, joita voisi perustellusti nimittää NoSQL-tietokannoiksi. Esimerkiksi Lotus Notesia, joka kehitettiin alun perin vuonna 1989, voidaan luonnehtia dokumenttivarastoksi. Mohanin mukaan sen ja nykyisten NoSQL-tietokantojen arkkitehtuurit muistuttavat toisiaan. MUMPS-ohjelmointikieli, joka kehitettiin 1960-luvulla terveydenhuollon tietojärjestelmiä varten, on samalla myös avain-arvo-varasto. MUMPS:in pohjalta on kehitetty myös GT.M, joka on avoimen lähdekoodin avain-arvo-varasto [Mohan, 2013].

NoSQL-tietokannat ovat yleistyneet erityisesti internetissä toimivissa palveluissa, joissa niiden ominaisuudet ovat erityisen hyödyllisiä. Internet-palveluille erityisen tärkeitä ominaisuuksia tietokantajärjestelmässä ovat joustavuus, tehokkuus ja saavutettavuus. NoSQL-tietokannat mahdollistavat tietokannan rakenteen joustavamman muokkaamisen. Relaatiotietokannoissa tietokannan rakenteen muokkaaminen aiheuttaa aina käyttökatkon. NoSQL-tietokantaa käyttämällä voi olla mahdollista lisätä palveluun uusia ominaisuuksia ilman käyttökatkoa. Web-palvelujen on myös selvittävä suuresta määrästä luku- ja kirjoitusoperaatioita ilman huomattavaa viivettä. Relaatiotietokantojen ominaisuudet, kuten liitosoperaatiot ja lukitukset heikentävät suorituskykyä erityisesti hajautetuissa tietokantajärjestelmissä. Saavutettavuuden takia tietokantajärjestelmän tulee luoda datasta kopioita useille eri palvelimille ja siinä tulee olla mekanismeja, joilla varaudutaan vikatilanteisiin. Järjestelmän on myös osattava varautua yllättäviin suuriin kävijämääriin jakamalla kuormaa eri palvelinten kesken. Koska relaatiotietokannoissa painotetaan lähtökohtaisesti eheyttä saavutettavuuden kustannuksella, NoSQL-tietokanta saattaa olla parempi vaihtoehto järjestelmän saavutettavuuden kannalta. [Hecht and Jablonski, 2011]

Historiallisesti relaatiotietokannat ovat olleet suosituimpia tietovarastoja kaikenlaisissa sovelluksissa ja niiden asema on vieläkin vahva [Atzeni *et al.*, 2013]. Relaatiotietokantojen tulevaisuudesta on hyvin erilaisia näkemyksiä. Stonebraker ja muut [2007] pitävät relaatiotietokantoja aikansa tuotoksina, jotka eivät toimi optimaalisesti nykyaikana, kun laitteistot ja tarpeet ovat hyvin erilaisia kuin 70-luvulla, kun relaatiotietokantoja kehitettiin. Mohan [2013] taas uskoo relaatiotietokantojen olevan edelleen paras ratkaisu moneen tarpeeseen ja kritisoi useita NoSQL-tietokantoja niiden hätiköidyistä suunnitteluratkaisuista. Hänen mukaansa useita NoSQL-tietokantojen ominaisuuksia on kokeiltu erilaisissa järjestelmissä jo vuosikymmeniä sitten, huonolla menestyksellä.

Relaatiotietokantojen hyvänä puolena pidetään erityisesti SQL-kieltä [Atzeni *et al.,* 2013]. SQL on sekä helposti luettava että ilmaisuvoimainen kyselykieli. Toisaalta SQL on ehkä liiankin laaja ja ilmaisuvoimainen sopiakseen yhtä hyvin kaikkiin käyttötarkoituksiin. Stonebraker ja muut [2007] ehdottavat uusien kyselykielten kehittämistä eri sovellusalueille sen mukaan, mitä SQL:n ominaisuuksia kyseisellä sovellusalueella todella

tarvitaan. Ominaisuuksien karsiminen toisi mahdollisuuksia optimointiin ja siten lisää tehokkuutta.

Toinen relaatiotietokantojen hyvä puoli on se, että tietomalli ja fyysinen tietokanta voidaan pitää erillään [Atzeni *et al*, 2013]. On olemassa vakiintuneet toimintatavat, miten sovellusalueen entiteetit ja suhteet muunnetaan relaatiotietokannaksi, ja tietomallia voi suunnitella välittämättä teknisistä yksityiskohdista. Atzenin ja muiden [2013] mukaan tämä ei ole lainkaan yhtä selkeää ja vakiintunutta NoSQL-tietokantoja käytettäessä. Tähän tietysti vaikuttaa sekin, että NoSQL-tietokannat ovat kovin erilaisia keskenään, eikä kaikille niille sopivaa toimintamallia ole helppo kehittää.

Relaatiotietokantojen huonona puolena voidaan puolestaan pitää sitä, että tietokannan skeeman muuttaminen käytössä olevassa tietokannassa on työlästä. NoSQL-tietokannoissa tietokannan rakenteen muuttaminen sen sijaan on useimmiten dynaamista, koska skeema puuttuu kokonaan tai sen määrittely on löyhempää kuin relaatiotietokannoissa. Lisäksi relaatiomalli poikkeaa semanttisesti esimerkiksi oliomallista, ja sen vuoksi relaatiotietokantaa käytettäessä tapa, jolla sama entiteetti on esitetty tietokannassa ja ohjelmakoodissa on erilainen. Ainakin joissain NoSQL-tietokannoissa datan esitystapa on lähempänä ohjelmointikielen esitystapaa. [Atzeni *et al.*, 2013]

SQL:n (ja samalla relaatiotietokantojen) etuna on SQL:n yhtenäinen standardi, jonka relaatiomallia noudattavat tietokannat toteuttavat ainakin periaatteessa, vaikka useita poikkeuksiakin on. SQL:n standardista on vuosikymmenten kuluessa tullut hyvin monimutkainen ja vaikeaselkoinen. Relaatiotietokantojen kehityksen ei myöskään koeta olevan akateemisen maailman tai niiden käyttäjien omissa käsissä, vaan niitä markkinoivien suurten yritysten hallinnassa. [Atzeni *et al*., 2013]

NoSQL-tietokantojen lisäksi suuria datamääriä käsittelevissä internet-sovelluksissa, joissa tehokkuus on tärkeää, voidaan käyttää niin kutsuttuja NewSQL-tietokantoja. NewSQL-tietokannat noudattavat relaatiomallia, mutta sallivat tietokannan osittamisen usealle eri palvelimelle. NewSQL-tietokannat tarjoavat monia perinteisten relaatiotietokantojen ominaisuuksista, kuten mahdollisuuden ACID-transaktioihin. Myös mahdollisuus käyttää SQL-kieltä on tärkeä ominaisuus NewSQL-tietokannoissa. Liitosoperaatiot, jotka vaativat taulujen yhdistämistä usean palvelimen välillä, ovat NewSQL-tietokannoissa erittäin tehottomia, joten jotkut niistä rajoittavat käyttäjän mahdollisuuksia tehdä tällaisia liitosoperaatioita. [Cattell, 2010]

3.3. Aggregointi ja liitosehdot NoSQL-tietokannoissa

NoSQL-tietokannat eivät useimmiten tarjoa yhtä ilmaisuvoimaista hakua kuin relaatiotietokannat ja SQL-kieli tarjoavat. Tämä asettaa rajoituksia sille, miten tietoa voidaan

NoSQL-tietokannoista hakea. Joissakin NoSQL-tietokannoissa näitä ominaisuuksia, kuten liitosoperaatioita on toteutettu, mutta niiden käyttöä ei suositella, koska se saattaa hidastaa tietokannan toimintaa. Puuttuvia ominaisuuksia voidaan korvata myös sovelluslogiikan avulla.

Rith ja muut [2014] ovat hahmotelleet, mitä SQL-kielen ominaisuuksia voidaan käyttää useimmissa NoSQL-tietokannoissa, mitä joissain NoSQL-tietokannoissa ja mitä taas ei juurikaan ole käytössä NoSQL-tietokannoissa. Kaikki NoSQL-tietokannat tukevat *CRUD* (create, read, update, delete) –toiminnallisuutta. Rivejä voidaan siis lisätä tietokantaan ja niitä voidaan päivittää, poistaa ja lukea. SQL:ssä tämä vastaa INSERT-, SELECT-, UPDATE- ja DELETE-lauseita tietyin rajoituksin.

Lisäyksiä voidaan tehdä vain niin, että annetaan lisättävälle riville yksikäsitteinen avain, ja arvot on annettava suoraan, kun taas SQL:ssä voidaan käyttää INSERT-lauseen sisäistä SELECT-lausetta. Rivin lukemisessa ei voida käyttää liitosoperaatioita eri taulujen välillä, eikä myöskään sisäisiä SELECT-lauseita, kuten SQL:ssä. Myöskään aggregaatiot eivät ole mahdollisia. Rivin päivittämisessä ja poistamisessa voi myöskin käyttää vain taulun sarakkeita, eikä alikyselyitä. Tällä tavoin rajoitettu CRUD-toiminnallisuus tarjotaan Rithin ja muiden [2014] mukaan kaikissa NoSQL-tietokannoissa.

Joissakin NoSQL-tietokannoissa voidaan käyttää SQL:n CREATE TABLE –tyyppisiä komentoja. Avain-arvo-varastoissa ne eivät toimi, koska varastot ovat skeemattomia, eikä tietotyyppejä arvoille voi siis määritellä. Monissa muissa NoSQL-tietokannoissa tietotyyppien määrittely on kuitenkin mahdollista.

Rithin ja muiden [2014] mukaan mitkään NoSQL-tietokannat eivät sellaisenaan tue transaktioita ainakaan niin, ettei tietokannan toiminnan tehokkuus kärsisi. Mikään NoSQL-tietokanta ei myöskään tue tietokantatriggereitä. Triggeri on tietokantajärjestelmään tallennettu ohjelma, joka määritellään suoritettavaksi ennen tai jälkeen tiettyyn tauluun kohdistuvan INSERT-, UPDATE- tai DELETE-komennon [Kroenke and Auer, 2016]. Jotkut tietokantajärjestelmät tukevat myös näkymiin liitettäviä INSTEAD OF-triggereitä ja skeeman määrittelykomentojen (esimerkiksi CREATE, ALTER ja DROP) yhteydessä suoritettavia triggereitä. Triggerien avulla voidaan muun muassa luoda sarakkeille oletusarvoja, luoda datalle rajoitteita, päivittää näkymiä ja huolehtia tietokannan eheydestä.

Sovelluslogiikalla voidaan pyrkiä parantamaan NoSQL-tietokantojen toimintaa. Tavoitteena voi olla esimerkiksi aggregointitietojen haun nopeuttaminen, koska joissain NoSQL-tietokannoissa aggregointitietojen hakeminen on hitaampaa kuin relaatiotietokannoissa [Parker *et al.*, 2013]. Toisaalta esimerkiksi sararakeperhevarastoissa aggregointitietojen hakeminen on erityisen nopeaa vertikaalisen osittamisen takia. Pasqualin ja muiden [2016] ratkaisussa dokumenttivaraston dokumentteihin indeksoidaan valmiiksi

kenttiä, jotka sisältävät aggregointitietoja dokumentin alidokumenteista. Tämä mahdollistaa nopeamman aggregointitietojen haun ja tällä tavoin on mahdollista myös lisätä haun ilmaisuvoimaa, kun alidokumenteista jokin voidaan tietyn ominaisuuden perusteella nostaa päätasolle ja siihen voidaan suoraan kohdistaa hakuja. Ratkaisun heikkoutena on se, että tarvittavat alidokumenttien ominaisuudet on tunnettava hyvin etukäteen.

Sovelluslogiikalla voidaan myös helpottaa NoSQL-tietokantojen käyttöönottoa rakentamalla yhteisiä rajapintoja useammalle NoSQL-tietokannalle. Moni NoSQL-tietokanta tarjoaa MapReduce-rajapinnan, jolla pystyy suodattamaan tietokannan rivejä ja suorittamaan toimintoja suodatetun tulosjoukon alkioille. MapReduce perustuu varhaiseen funktionaaliseen ohjelmointiin [Dean and Ghemawat, 2008]. MapReducessa ohjelman toiminta määritellään **map**- ja **reduce**-funktioiden avulla. Näitä funktioita käytetään kuitenkin MapReduce-rajapinnassa hieman eri merkityksessä kuin funktionaalisessa ohjelmoinnissa. MapReduce on erittäin tehokas menetelmä isojen datajoukkojen käsittelyyn rinnakkaisesti. MapReducen ilmaisuvoima vastaa SQL:n ilmaisuvoimaa [Chattopadhyay *et al.*, 2011], joten sitä käyttämällä NoSQL-haku tulee paljon monipuolisemmaksi. MapReducen käyttö vie kuitenkin osan NoSQL-tietokantojen tehokkuudesta [Rith *et al.*, 2014].

Rith ja muut [2014] ovat kehittäneet rajapinnan, jolla kahteen eri NoSQL-tietokantaan voi tehdä SQL-kyselyitä ilman MapReducen käyttämistä. Myös Atzeni ja muut [2012] ovat tehneet Javalle rajapinnan, jolla pystyy tekemään luku-, kirjoitus- ja poisto-operaatioita kolmeen eri tyyppiseen NoSQL-tietokantaan (HBase, Redis ja MongoDB). Rivejä, avain-arvo-dataa tai dokumentteja käsitellään Javan olioina, jotka muunnetaan tietokantaan tallentamista varten ensin JSON-olioiksi, minkä jälkeen ne muunnetaan kunkin tietokannan käyttämään tietorakenteeseen.

3.4. Elasticsearch

Elasticsearch on erityisesti kokotekstin hakuun laajasta aineistosta tarkoitettu avoimen lähdekoodin hakukone [Kononenko *et al.*, 2014]. Elasticsearch perustuu Apache Lucene –hakukoneeseen. Lucene on Javalla kirjoitettu tiedonhakukirjasto, jossa on työkaluja muun muassa käänteisindeksin luomiseen [Gao *et al*., 2011].

Elasticsearch	**Relaatiotietokannat**
Indeksi	Tietokanta
Kuvaus (mapping)	Skeema
Dokumentin tyyppi	Taulu
Dokumentti	Rivi

Taulukko 2. Käsitteiden vastaavuuksia Elasticsearch-hakukoneessa ja relaatiotietokannoissa. [Kononenko *et al.*, 2014]

Elasticsearchia voi käyttää myös NoSQL-tietokantana, koska sillä on kaikki olennaisimmat dokumenttivaraston ominaisuudet. Elasticsearchissa dokumentit tallennetaan JSON-muodossa. Myös dokumentin kuvaus ja kyselyt annetaan JSON-muodossa. Dokumentin kentät voivat sisältää alidokumentteja, listoja ja alidokumenttilistoja. Elasticsearchissa dokumentteja voi lisätä, päivittää ja poistaa indeksistä. Elasticsearchia voi käyttää joko skeemattomana tai kuvauksen avulla määritellä, mitä avaimia eri dokumenttityypeillä voi olla.

Taulukossa 2 on listattu Kononenkon ja muiden [2014] hahmottelemia vastaavuuksia Elasticsearchin käsitteiden ja relaatiotietokantojen elementtien välillä. Elasticsearchin indeksi vastaa relaatiotietokantajärjestelmän tietokantaa. Indeksi on nimiavaruus, joka sisältää indeksin dokumenttityyppien kuvaukset. [Elastic, 2017c] Indeksi voi sisältää useita eri tyyppisiä dokumentteja ja myös haku- ja päivityskyselyt kohdistetaan indeksiin, kuten relaatiotietokannassa tietokanta sisältää tauluja ja kysely kohdistetaan yhden tai useamman tietokannan tauluihin.

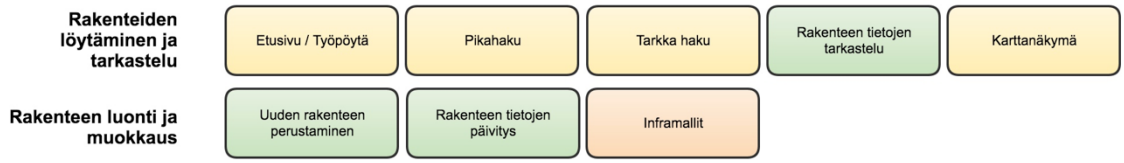
Kuvauksella määritellään Elasticsearchissa dokumenttien ja niiden sisältämien kenttien tyypit. Kuvauksessa määritellään myös, miten eri kentät varastoidaan, analysoidaan ja indeksoidaan. Kuvaus vastaa siis relaatiotietokantojen skeemaa.

4. Taitorakennerekisteri

Tässä luvussa esitellään Liikenneviraston Taitorakennerekisteri-tietojärjestelmä. Luvussa käydään läpi järjestelmän tärkeimmät toiminnallisuudet ja luodaan katsaus sen tietomalliin ja arkkitehtuuriin. Erityisesti keskitytään Taitorakennerekisterin hakutoiminnallisuuteen, joka on tämän tutkimuksen kannalta järjestelmän olennaisin osa. Luvussa esitellään myös järjestelmän haun rajoitteet alidokumentteihin kohdistuvissa hakutarpeissa ja tarpeet hakua laajentavalle toiminnallisuudelle.

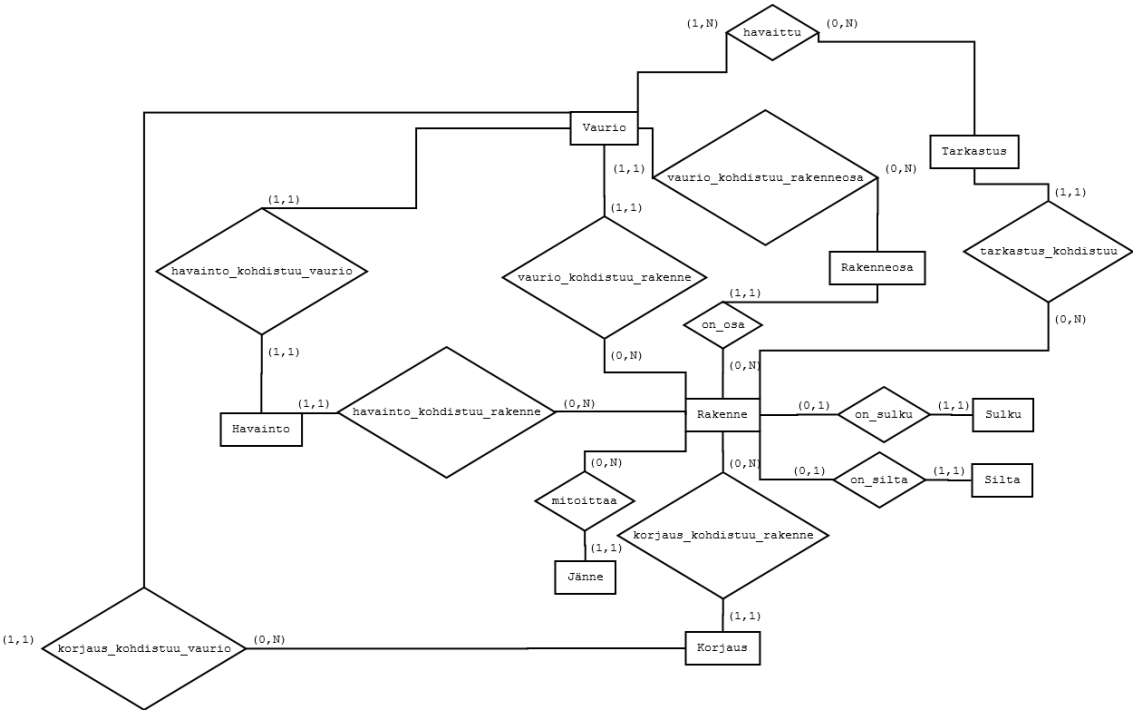
4.1. Taitorakennerekisterin tavoitteet ja tietomalli

Taitorakennerekisteri on Liikenneviraston omistama ja Solita Oy:n toimittama tietojärjestelmä, jolla hallitaan tietoja taitorakenteista ja niiden tarkastuksista. Liikennevirasto [2013] määrittelee taitorakenteen sellaiseksi rakenteeksi, jonka rakentamiseksi on laadittava lujuuslaskelmiin perustuvat suunnitelmat ja jonka rakenteellinen vaurioituminen suunnittelu- tai rakennusvirheen seurauksena saattaa aiheuttaa vaaraa ihmisille tai liikennejärjestelmälle ja/tai merkittäviä korjauskustannuksia rakenteelle tai sen välittömälle ympäristölle. Taitorakenteita ovat esimerkiksi sillat, tunnelit ja sulut. Taitorakennerekisterissä ylläpidetään sekä Liikenneviraston että muiden tahojen omistamien taitorakenteiden tietoja ja sen avulla tuotetaan myös raportteja ja tilastoja taitorakenteista. Taitorakennerekisteriin tallennetaan tiedot taitorakenteiden tarkastuksista ja korjauksista. Järjestelmän käyttäjiä ovat muun muassa taitorakenteiden tarkastuksia tekevien yritysten työntekijät ja Liikenneviraston työntekijät, jotka tarvitsevat tietoja ja tilastoja taitorakenteista.



Kuva 1. Osa Taitorakennerekisterin toiminnallisista kokonaisuuksista. [Solita Oy, 2015]

Kuvassa 1 on esitelty esimerkkinä kaksi Taitorakennerekisterin toiminnallista kokonaisuutta. Kuvassa on käytetty myös värejä kuvaamaan osa-alueita, joille toiminnallisuudet kuuluvat. Keltaisella merkityt toiminnallisuudet kuuluvat tiedon hakemisen ja organisoinnin osa-alueelle, vihreällä merkityt rakenteen käsittelyn osa-alueelle ja oranssilla merkitty toiminnallisuus kuuluu muualla tuotetun materiaalin hallinnan osa-alueelle. Kuvassa esiteltyjen käyttötapausten lisäksi tärkeitä toiminnallisuuksia järjestelmässä ovat tarkastusten ja korjausten lisääminen järjestelmään. Taitorakennerekisteriä käytetään selaimen kautta, joten sen käyttö onnistuu myös esimerkiksi tablettilaitteella suoraan tarkastuspaikalta.



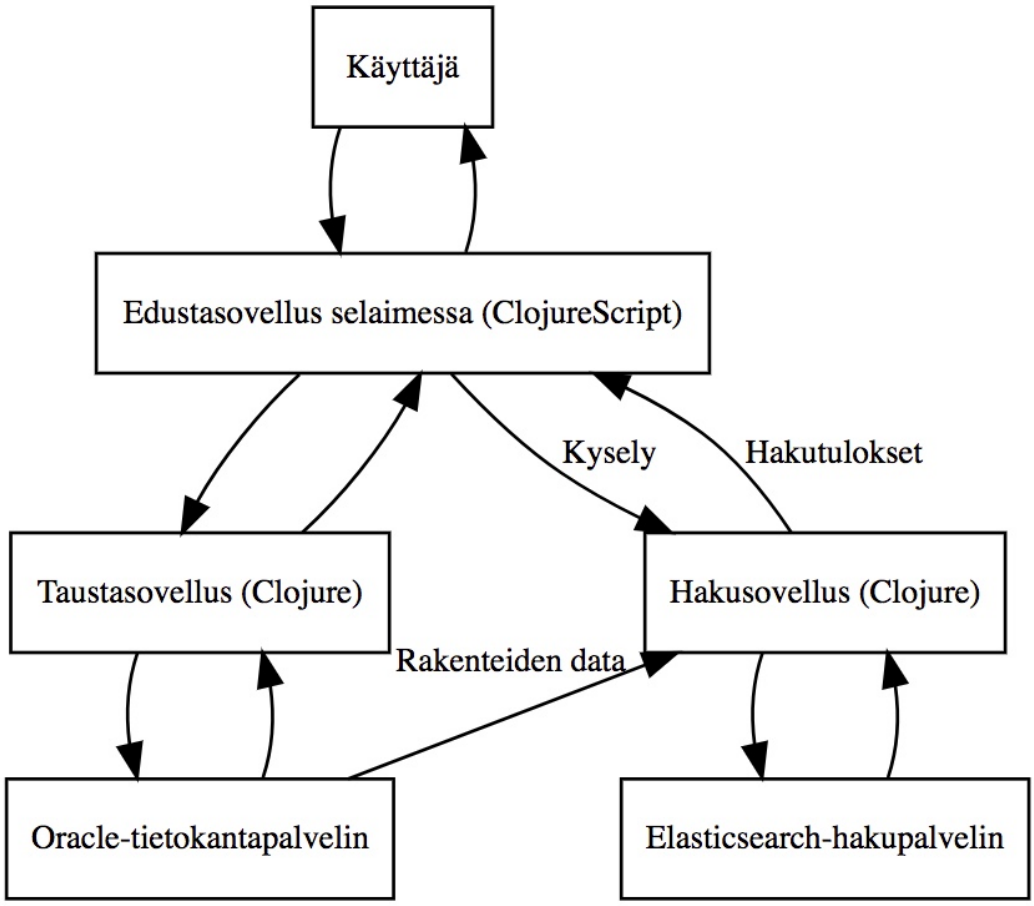
Kaavio 1. Esimerkki Taitorakennerekisterin tietomallista ER-kaaviona.

Kaaviossa 1 esitellään esimerkkinä osa Taitorakennerekisterin tietomallista ER-kaaviona. Kaaviosta on yksinkertaisuuden vuoksi jätetty pois attribuutit. Kaikkien rakenteiden perustiedot ovat rakenne-taulussa, ja eri *rakenneluokkiin* kuuluvat rakenteet on esitetty tauluin, joista viitataan rakenne-tauluun, mistä on esimerkkinä silta- ja sulku-taulut. Nämä taulut sisältävät rakenteiden rakenneluokkakohtaiset lisätiedot. Rakenneosa ja jänne liittyvät aina yhteen rakenteeseen, samoin tarkastus ja korjaus. Vaurio puolestaan liittyy rakenteeseen ja rakenneosaan, jossa se on havaittu. Vaurio voi liittyä myös korjaukseen, jossa vaurio on korjattu. Havainto liittää yhteen vaurion ja tarkastuksen, jossa kyseinen vaurio on havaittu.

Taitorakennerekisterin palvelinsovelluksen ohjelmisto on toteutettu Clojure-ohjelmointikielellä. Valintaa perustellaan [Solita Oy, 2014] sillä, että Clojure on moderni kieli, joka sopii hyvin ylläpidettävyydeltään pitkän elinkaaren sovellukseen. Se on kuitenkin jo laajasti käytössä ja kypsä kieli, joten sille löytyy paljon kirjastoja erilaisiin tarpeisiin. Clojuressa voidaan käyttää myös kaikkia Javalle tehtyjä kirjastoja. Clojure on funktionaalinen ohjelmointikieli, jota käännetään Javan virtuaalikoneen tavukoodiksi [Clojure.org, 2017]. Taitorakennerekisterin selainsovellus on toteutettu ClojureScript-ohjelmointikielellä, joka muistuttaa hyvin paljon Clojurea, mutta jota käännetään JavaScript-kieleksi, jota selain suorittaa [ClojureScript.org, 2017]. Jonkin verran ohjelmakoodia on myös molempien sovellusten yhteisessä käytössä.

4.2. Haku Taitorakennerekisterissä

Kaaviossa 2 kuvataan Taitorakennerekisterin haun arkkitehtuuri. Käyttäjä kommunikoi ClojureScript-kielellä toteutetun edustasovelluksen kanssa, joka puolestaan on yhteydessä Clojurella toteutettuihin taustasovellukseen ja hakusovellukseen. Taustasovellus huolehtii rakenteiden tietojen näyttämisestä ja päivittämisestä. Kun rakenteen tietoihin tulee päivitys, se lisätään indeksointi-nimiseen tauluun, josta hakusovellus ajastettuna hakee uudelleenindeksoitavat rakenteet ja indeksoi ne uudelleen Elasticsearch-hakukoneeseen. Käyttäjän tehdessä kyselyn edustasovelluksessa, se lähetetään hakusovellukselle, joka välittää kyselyn Elasticsearch-palvelimelle. Elasticsearch suorittaa haun ja palauttaa hakutulokset hakusovellukselle. Hakutulokset välitetään edelleen edustasovellukselle, joka näyttää ne käyttäjälle käyttöliittymässä.



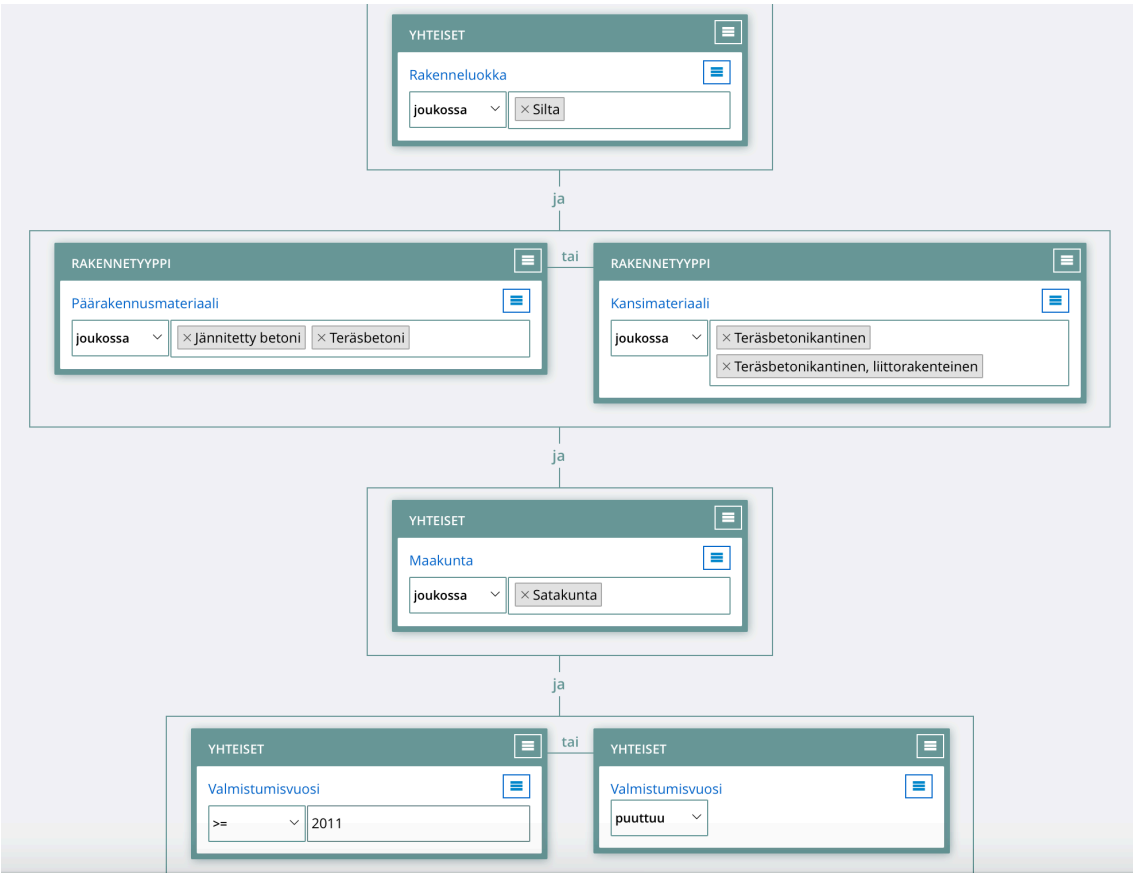
Kaavio 2. Taitorakennerekisterin haun arkkitehtuuri. [Solita Oy, 2016]

Taitorakennerekisterissä Elasticsearchin indeksin dokumenttityyppi on rakenne. Kuvaus on Taitorakennerekisterin Elasticsearch-indeksissä täsmällinen. Kaikki kentät, joita rakenteella voi olla, on määritelty etukäteen. Tiedon esitys haussa poikkeaa hieman tietokannan rakenteesta. Tietokannassa on kaavion 1 esittämällä tavalla kuvattu rakenne siten, että sen perustiedot ovat rakenne-taulussa ja rakenneluokkakohtaiset lisätiedot sekä

rakenteeseen liittyvät muut käsitteet, kuten rakenneosat ja tarkastukset ovat omissa tauluissaan. Haussa taas rakenne on yksi kokonainen dokumentti, jossa useita alikäsitteiden tietoja on tuotu suoraan päätasolle haun ja hakutulosten esittämisen yksinkertaistamiseksi. Rakenteella on myös useita listatyyppisiä arvoja, jotka sisältävät alidokumentteja. Näitä ovat esimerkiksi jänne, rakenneosa ja tarkastus. Nämä arvot haetaan tauluista, joissa rakenteen tunniste on vierasavaimena. Kaikkia rakenteen tietoja ei tuoda lainkaan hakuun. Se, mitä rakenteen tietoja ja missä muodossa relaatiotietokannasta tuodaan hakuindeksiin, on muotoutunut käyttäjien tarpeiden ja palautteen perusteella ja siihen on tuotu lisää tietoja järjestelmän iteratiivisen kehitystyön myötä.

Vuorovaikutus Clojure-sovelluksen ja Elasticsearch-palvelimen välillä tapahtuu Taitorakennerekisterissä Elastisch-asiakaskirjaston avulla. Sekä kyselyt että indeksin päivitys tehdään Elastischin kautta. Rakenteiden tiedot Elasticsearchin indeksiin tuodaan Oracle-tietokannasta, josta myös haetaan suurin osa käyttöliittymässä esitettävistä tiedoista. Jos tietoja muokataan tietokannassa, indeksoidaan muuttuva rakenne Oraclen tietokantatriggerin avulla kokonaan uudelleen.

Kysely Elasticsearchille muodostetaan käyttäjän käyttöliittymässä tekemän haun perusteella. Taitorakennerekisterissä on kaksi erilaista hakutoiminnallisuutta. Toinen on pikahaku, jossa käyttäjä kirjoittaa haettavan merkkijonon tekstikenttään. Merkkijonoa käsitellään yhtenä hakusanana, jota haetaan tekstihaulla rakenteen eri kentistä. Hakusanasta muodostetaan Elasticsearchin *sovellusaluekyselykielen* (domain specific query language) kysely, joka etsii hakusanaa rakenteen nimestä, tunnuksesta ja kuvauksesta ja rakenteen omistajan, kunnossapitäjän ja sijaintikunnan nimistä. Eri kenttiä painotetaan haussa eri tavalla. Esimerkiksi rakenteita, joiden tunnuksessa hakusana esiintyy, painotetaan hakutuloksessa enemmän kuin rakenteita, joilla hakusana esiintyy kuvauksessa. Pikahaku ei tällä hetkellä tue monimutkaisempia kyselyrakenteita. Toinen hakutoiminnallisuus on laaja haku, jossa käyttäjä muodostaa kyselyn graafisen käyttöliittymän avulla. Laajan haun avulla käyttäjän on mahdollista tehdä kyselyitä, jotka kohdistuvat rakenteen tiettyihin kenttiin ja käyttävät Boolen JA- ja TAI-operaattoreita.



Kuva 2. Taitorakennerekisterin laajan haun käyttöliittymä.

Kuvassa 2 on esimerkki kyselystä laajan haun graafisessa käyttöliittymässä. Kyselyllä haetaan sillat, joiden päärakennusmateriaali on jännitetty betoni tai jotka ovat teräsbetonikantisia tai teräsbetonikantisia ja liittorakenteisia, jotka sijaitsevat Satakunnassa ja jotka ovat valmistuneet vuonna 2011 tai sen jälkeen tai joiden tiedoista puuttuu valmistumisvuosi. Laajan haun tarkoitus on tarjota työkalu monipuolisiin hakuihin niin, että Elasticsearchin hyödyt, kuten nopeus, säilytetään.

Järjestelmässä on myös sisäinen esitys kyselylle, mikä helpottaa kyselyn kokoamista käyttäjän syötteen perusteella. Kysely muunnetaan sekä käyttöliittymässä esitettäväksi kyselyn tekstiesitykseksi että Elasticsearch-kyselyksi. Esimerkissä 1 on kysely Taitorakennerekisterin sisäisessä esitysmuodossa. Kyselyssä etsitään Tampereen museorakenteet.

```
{:kohde :rakenne,
 :rivit
 [{:sarakkeet
   [{:lausekkeet
     [{:tyyppi :listan-jokin,
       :hakupolku "yhteiset",
       :sarakkeet
       [{:lausekkeet
         [{:tyyppi :yksiarvoinen,
           :hakupolku "historiallinenmerkittavyys",
           :operaattori :joukossa,
           :verrattava
           [{:tunnus "13",
             :nimi "Museorakenne",
             :inaktiivinen false}]}]}]}]}]
  {:sarakkeet
   [{:lausekkeet
     [{:tyyppi :listan-jokin,
       :hakupolku "yhteiset",
       :sarakkeet
       [{:lausekkeet
         [{:tyyppi :yksiarvoinen,
           :hakupolku "sijaintikunta",
           :operaattori :joukossa,
           :verrattava ({:tunnus "837", :nimi "Tampere"})}]}]}]}]}]}
```

Esimerkki 1. Kysely Taitorakennerekisterin sisäisessä esitysmuodossa.

4.2.1. Elasticsearch Taitorakennerekisterissä

Esimerkissä 2 on osa Taitorakennerekisterin rakenteen Elasticsearch-kuvausta. Esimerkin kuvauksessa rakenteella voi olla kentät jänne, kuvaus, nimi, tarkastus, korjaus, valmistumisvuosi ja vaurio. Näistä esimerkiksi jänne-kenttä koostuu listasta jänne-tyyppisiä alidokumentteja, joiden sallitut kentät on myös määritelty kuvauksessa. Jokaiselle kentälle on määritelty myös tyyppi. Esimerkiksi valmistumisvuosi on tyyppiä short, joka on yksi Javan kokonaislukutyypeistä.

```
{"taitorakennerekisteri": {
      "mappings": {
         "rakenne": {
            "dynamic": "strict",
            "properties": {
               "janne": {
                  "type": "nested",
                  "properties": {
                     "jannetyyppi": {
                        "type": "string",
                        "index": "not_analyzed"},
                     "pituus": {
                        "type": "float"}},
               "korjaus": {
                  "type": "nested",
                  "properties": {
                     "kokonaiskustannukset": {
                        "type": "integer"},
                     "korjaustila": {
                        "type": "string",
                        "index": "not_analyzed"},
                     "korjaustoimenpide": {
                        "type": "string",
                        "index": "not_analyzed"},
                     "korjaustyyppi": {
                        "type": "string",
                        "index": "not_analyzed"},
                     "paivamaara": {
                        "type": "date",
                        "format": "dd.MM.yyyy"}}},
               "kuvaus": {
                  "type": "string",
                  "store": true,
                  "analyzer": "keyword_lowercase"},
               "nimi": {
                  "type": "string",
                  "store": true,
                  "analyzer": "keyword_lowercase"},
               "tarkastus": {
                  "type": "nested",
                  "properties": {
                     "kommentitjapuutteet": {
                        "type": "string",
                        "store": true,
                        "analyzer": "keyword_lowercase"},
                     "paivamaara": {
                        "type": "date",
                        "format": "dd.MM.yyyy"},
                     "tarkastustyyppi": {
                        "type": "string",
                        "index": "not_analyzed"}},
```

```
"valmistumisvuosi": {
   "type": "short"},
"vaurio": {
   "type": "nested",
   "properties": {
      "laajuus": {
         "type": "float"},
      "rakenneosatyyppi": {
         "type": "string",
         "index": "not_analyzed"},
      "vauriotyyppi": {
         "type": "string",
         "index": "not_analyzed"}}}}}}
```

Esimerkki 2. Osa kuvausta Elasticsearchissa.

Dokumentin tyyppi määrittelee, mitä kenttiä minkäkin tyyppisellä dokumentilla voi olla. Elasticsearchissa kaikilla dokumenteilla ei kuitenkaan välttämättä ole kaikkia tyypissä määriteltyjä kenttiä [Kononenko *et al.*, 2014]. Elasticsearchissa voidaan käyttää joko dynaamista tai täsmällistä kuvausta. Indeksoitaessa uusi dokumentti, jossa on kenttä, jota ei ole määritelty kuvauksessa, jos käytössä on dynaaminen kuvaus, Elasticsearch päättelee kentän arvon perusteella kentän tyypin ja lisää sen kuvaukseen. Tiukassa kuvauksessa taas tällainen dokumentti aiheuttaa poikkeuksen, eikä sitä indeksoida. Voidaan käyttää myös vaihtoehtoa, jossa kentät, joita ei ole kuvauksessa, jätetään huomiotta ja dokumentti indeksoidaan ilman niitä [Elastic, 2016]. Esimerkin 2 kuvaus on määritelty tiukaksi avainsanoilla **dynamic: strict**. Annetuista sallituista kentistä poikkeavan dokumentin tuominen indeksiin aiheuttaa siis poikkeuksen.

```
<indeksi> ::= {"<indeksin-nimi>": {<kuvaus>}}
<kuvaus> ::= "mappings": {<dokumenttityyppi>}
<dokumenttityyppi> ::= "<dokumenttityypin-nimi>": {<dokumenttityypin-kuvauksen-
                       dynaamisuus> <ominaisuudet>}
                       | <dokumenttityypin-nimi>": {<dokumenttityypin-kuvauksen-dynaamisuus>
                         <ominaisuudet>}, <dokumenttityyppi>
<dokumenttityypin-kuvauksen-dynaamisuus> ::= , "dynamic": <dokumenttityypin-kuvauksen-dynaa-
                                             misuuden-arvo> | ""
<dokumenttityypin-kuvauksen-dynaamisuuden-arvo> ::= true | false | "strict"
<ominaisuudet> ::= "properties": {<ominaisuus>}
<ominaisuus> ::= "<ominaisuuden-nimi>": {<tyyppi> <ominaisuudet> <analyysi> <formaatti> <tal-
                 lenna>}
                 | "<ominaisuuden-nimi>": {<tyyppi> <ominaisuudet> <analyysi> <formaatti>
                 <tallenna>} , <ominaisuus>
<tyyppi> ::= , "type": <tyypin-arvo>, <tyypin-arvo> ::= "text" | "keyword" | "long" | "integer"
           | "short" | "byte" | "double" | "float" | "date" | "boolean" | "binary" | "in-
           teger_range" | "float_range" | "long_range" | "double_range" | "date_range" |
           "object" | "nested" | "geo_point" | "geo_shape" | "ip" | "completion" | "to-
           ken_count" | "mapper-murmur3" | "attachment"
<analyysi> ::= , "index": "not_analyzed" | , "analyzer": <analysoijan-nimi> | ""
<formaatti> ::= , "format": <päivämääräformaatti> | ""
<tallenna> ::= , "store": <tallenna-arvo> | ""
<tallenna-arvo> ::= true | false
```

Esimerkki 3. Elasticsearchin kuvaus BNF-muodossa.

Esimerkissä 3 on esitetty Elasticsearchin kuvaus BNF-tyyppisesti siten, että terminaalit esitetään lihavoituna ja nonterminaalit kulmasulkujen välissä. Esimerkki ei kata kaikkia parametreja, joita kuvaukselle on mahdollista antaa, vaan sisältää esimerkin 2 parametrit. Parametrien kohdalla on kuitenkin käsitelty kaikki niille sallitut arvot. Jotkin määritelmistä ovat rekursiivisia. Kuvaus voi sisältää yhden tai useamman dokumenttityypin ja ominaisuudet voivat sisältää yhden tai useamman ominaisuuden. Indeksin nimi, dokumenttityypin nimi, ominaisuuden nimi, analysoijan nimi ja päivämääräformaatti on jätetty määrittelemättä tässä, koska käyttäjä voi määritellä ne itse haluamallaan tavalla. Analyysi kuvaa sitä, miten tekstimuotoisen ominaisuuden arvo tuodaan indeksiin. **Index: not analyzed** tarkoittaa, että arvo viedään sellaisenaan. **Analyzer** -avainsanalla voi antaa jonkin valmiin tai itse toteutetun analysoijan nimen. Analysoija voi esimerkiksi *stopata* (poistaa sanat, joilla ei ole haussa merkitystä) ja *stemmata* (muodostaa sanoista sanavartalot) tekstin jonkin tietyn luonnollisen kielen mukaan.

Elasticsearchin dokumentti vastaa relaatiotietokantojen riviä. Elasticsearchissa dokumentti voi kuitenkin olla hierarkkinen, eli sen kentät voivat sisältää muita dokumentteja. Relaatiotietokannoissa hierarkkisuutta ei yleensä ole, vaan rivit yhdistetään toisiin tietokannan riveihin liitosoperaatioiden avulla. Tehtäessä kysely Elasticsearchin indeksiin vastauksena palautuu kokoelma dokumentteja, jotka täyttävät kyselyn ehdot. Samoin päivitys- ja indeksointioperaatiot kohdistuvat aina dokumenttiin. [Kononenko *et al.*, 2014]

Elasticsearch soveltuu erityisen hyvin hakuun suuresta määrästä dataa, koska Elasticsearch osittaa dokumentit useille eri *sirpaleille* (shard). Jokaista sirpaletta voidaan käyttää rinnakkain, mikä tehostaa hakua [Kononenko *et al.*, 2014]. Tämän lisäksi myös sirpale jaetaan eri palvelimille, jos Elasticsearchin käytössä on useita eri palvelimia.

Elasticsearch tarjoaa mahdollisuuden tehdä kyselyitä joko REST-rajapinnan kautta tai joillekin ohjelmointikielille tarjolla olevien kirjastojen kautta. REST-rajapinnan kautta hakukonetta käytettäessä asiakasohjelma tekee kyselyn Elasticsearchin palvelimelle JSON-muodossa HTTP-pyyntönä, ja saa vastauksena hakutulokset myös JSON-muodossa.

Esimerkissä 4 on kysely Taitorakennerekisterin indeksistä. Ensimmäisenä on annettu indeksin nimi (taitorakennerekisteri) ja seuraavana toiminto (haku, **_search**). Sitten seuraa kysely JSON-muodossa. **_source**-avainsanalla määritellään, mitkä kentät hakutulokseen otetaan mukaan. Tämä vastaa SQL:n SELECT-lauseen osaa, jossa valitaan tulokseen mukaan otettavat sarakkeet. **Query**-avainsana vastaa SQL-kielen lauseen WHERE-osuutta, jossa annetaan hakutuloksen rajaavat ehdot. Query-osiossa voi käyttää useita eri tyyppisiä kyselyitä, kuten **bool** (Boolen logiikkaoperaattoreita sisältävä kysely), **range** (kentän arvon tulee olla tietyltä arvoalueelta) tai **wildcard** (tekstikenttään kohdistuva,

jossa voi käyttää jokerimerkkiä). **Term-**kyselyllä haetaan dokumentit, joiden annetun kentän arvo vastaa täysin annettua arvoa. **Sort**-avainsanalla kerrotaan, minkä kentän mukaan hakutulokset järjestetään. Sort vastaa SQL:n kyselyn ORDER BY-osuutta. Esimerkissä 5 on esimerkkiä 4 vastaava kysely SQL-kielellä.

Esimerkissä 6 näkyy osa esimerkin 4 kyselyn hakutuloksista. Alussa on tietoja kyselyn käyttämästä ajasta ja resursseista. **Hits**-osiossa on ensin hakutulosten määrä ja sitten hakutulokset JSON-olioina listassa. Vaikka kyselyssä haluttiin mukaan kenttä kuvaus, sitä ei ole ensimmäisellä hakutuloksella, koska Elasticsearch sallii dokumenttityypissä määritellyn kentän puuttumisen dokumentilta.

```
GET taitorakennerekisteri/_search
{"_source": ["nimi","valmistumisvuosi", "kuvaus"],
   "query": {"term": {"valmistumisvuosi": {"value": 1999}}},
   "sort": [{"nimi": {"order": "asc"}}]}
```

Esimerkki 4. Haku Elasticsearch-hakukoneesta sovellusaluekyselykielellä.

```
SELECT nimi, valmistumisvuosi, kuvaus
FROM rakenne
WHERE valmistumisvuosi = 1999
ORDER BY nimi ASC;
```

Esimerkki 5. Esimerkkiä 4 vastaava kysely SQL-kielellä.

```
{"took": 40, "timed_out": false,
   "_shards": {"total": 5, "successful": 5, "failed": 0},
   "hits": {"total": 220, "max_score": null,
      "hits": [{"_index": "taitorakennerekisteri",
            "_type": "rakenne",
            "_id": "7651",
            "_score": null,
            "_source": {
               "nimi": "Aaltosenpolun alikulkukäytävä",
               "valmistumisvuosi": 1999
            "sort": [
               "aaltosenpolun alikulkukäytävä"
            ]},
        {"_index": "taitorakennerekisteri",
            "_type": "rakenne",
            "_id": "23821",
```

Esimerkki 6. Osa hakutuloksista esimerkin 4 kyselyyn.

```
GET taitorakennerekisteri/_search
{"_source": ["nimi", tarkastus.paivamaara","tarkastus.kommentitjapuutteet"],
   "query": {nested": {"path": "tarkastus",
                       "query": {"wildcard": {"tarkastus.kommentitjapuutteet": {
                                              "value": "*puuttuu*"}}}}}}
```

Esimerkki 7. Alidokumenttiin kohdistuva kysely sovellusaluekyselykielellä.

```
SELECT rakenne.nimi, tarkastus.paivamaara, tarkastus.kommentitjapuutteet
FROM rakenne INNER JOIN tarkastus ON (rakenne.id = tarkastus.rakenneid)
WHERE tarkastus.kommentitjapuutteet LIKE ’%puuttuu%’;
```

Esimerkki 8. Esimerkin 7 kysely SQL-kielellä.

Kyselyt monimutkaistuvat, kun tehdään hakuja alidokumentteja sisältävistä kentistä [Kononenko, 2014]. Esimerkissä 7 haetaan **nested**-kyselyn avulla rakenteet, joilla on ainakin yksi tarkastus, jonka kommentit ja puutteet sisältävät merkkijonon ”puuttuu”. **Nested**-kyselyllä hakutulokseen otetaan aina ne dokumentit, joiden jokin alidokumentti täyttää annetun ehdon. Esimerkissä 8 on edellä mainittu kysely SQL-kielellä. Sovellusaluekyselykielen **nested**-rakenne siis mahdollistaa yksinkertaisia liitosoperaatioita vastaavat kyselyt, mutta tarkempien rajausten tekeminen ei ole sillä mahdollista.

```
GET taitorakennerekisteri/_search
{"query": {"bool": {"should":
                     [{"bool":
                        {"must": [{"nested": {
                                     "path": "yhteiset",
                                     "query": {
                                       "terms": {
                                         "yhteiset.sijaintikunta":
                                           ["211"]}}}},
                                  {"nested": {
                                     "path": "yhteiset",
                                     "query": {
                                       "range": {
                                         "yhteiset.valmistumisvuosi": {
                                           "gt": 2005}}}}}]}},
                        {"nested": {"path": "yhteiset",
                                    "query": {
                                      "range": {
                                        "yhteiset.valmistumisvuosi": {
                                          "gt": 2014}}}}}]}}}
```

Esimerkki 9. Bool-kysely sovellusaluekyselykielellä.

Elasticsearchin sovellusaluekyselykielessä on myös vastaavuus SQL:ssä käytetyille Boolen logiikkaoperattoreille [Kononenko, 2014]. **Bool**-kyselyn **must**, **should** ja **must_not** –avainsanat eivät suoraan vastaa Boolen operaattoreita AND, OR ja NOT, mutta niillä voi ilmaista loogisesti samat lauseet. Hakutulosten tulee täyttää **must**-lohkon sisältämät kyselyt. **Should**-lohkon sisältämien kyselyiden ehtojen ei ole pakko täyttyä, mutta ne vaikuttavat dokumentin pistemäärään hakutuloksissa. Jos kyselyssä on annettu vain yksi **should**-lohko, sen tulee toteutua hakutuloksissa [Elastic, 2017]. Esimerkin 9 kyselyssä on käytetty **bool**-rakennetta. Siinä haetaan rakenteet, joiden sijaintikunnan tunnus on 211 ja jotka ovat valmistuneet vuoden 2005 jälkeen tai jotka ovat valmistuneet vuoden 2014 jälkeen. Sama kysely SQL-kielellä on esimerkissä 10. SQL-kielellä kyselyssä joudutaan tekemään liitosoperaatio, koska rakenteella voi olla useita sijaintikuntia.

```
SELECT *
FROM rakenne r
INNER JOIN rakennekunta rk ON (r.id = rk.rakenneid)
WHERE r.valmistumisvuosi > 2014 OR
      (rk.kuntanumero = 211 AND
      r.valmistumisvuosi > 2005);
```

Esimerkki 10. Esimerkin 9 kysely SQL-kielellä.

```
PUT taitorakennerekisteri/rakenne/1
{
    "nimi": "Uusi rakenne",
    "valmistumisvuosi": 2017
}
```

Esimerkki 11. Dokumentin päivitys Elasticsearchissa.

Elasticsearchilla on mahdollista tehdä myös päivityksiä indeksin dokumentteihin. Esimerkissä 11 päivitetään rakenteen, jonka id on 1, tiedot annetun dokumentin mukaisiksi. Annetulla dokumentilla korvataan indeksissä oleva dokumentti, eikä päivityksiä yksittäisiin arvoihin voi tehdä. Elasticsearch tukee dokumenttien versiointia, joten optimistista lukitusta voi käyttää varmistamaan, että päivitys kohdistuu dokumentin oikeaan versioon [Elastic, 2016]. Taitorakennerekisterissä tietoja ylläpidetään Oracle-tietokannassa. Kun rakenteeseen tai johonkin siihen liittyvään käsitteeseen tulee muutoksia (tai jos lisätään kokonaan uusi rakenne), rakenteen tiedot indeksoidaan kokonaan uudelleen Elasticsearchin indeksiin PUT-komennolla, kuten esimerkissä 11.

4.3. Taitorakennerekisterin haun rajoitteet ja tarve lisätä ilmaisuvoimaa

Taitorakennerekisterin laajan haun käyttöliittymä mahdollistaa Boolen operaattorien käytön ja haun rakenteen sisältämistä alidokumenteista. Koska laajan haun käyttöliitty-

män on tarkoitus korvata suoraan tietokantaan tehtävät SQL-kyselyt, tarvitaan siinä kuitenkin myös ilmaisuvoimaisempia ominaisuuksia. Käyttäjän tiedontarpeena voi olla esimerkiksi löytää rakenteet, joiden laajin vaurio on kansilaatassa tai rakenteet, joiden viimeisin sukellustarkastus on tehty ennen vuotta 2000. Hakuehto kohdistuu siis tietyllä kriteerillä valittuun rakenteen alidokumenttiin.

Osa alidokumentteihin kohdistuvista tiedontarpeista tunnetaan tarkasti. Yllä olevan esimerkin viimeisin tarkastus on tunnettu hakutarve ja siksi viimeisimmän tarkastuksen tiedot tuodaan tarkastukset sisältävän listatyyppisen arvon lisäksi myös erikseen rakenteen päätasolle indeksointivaiheessa. Tämä muistuttaa Pasqualinin ja muiden [2016] toimintatapaa, jossa dokumentin päätasolle tuodaan alidokumentteja koskevaa aggregointitietoa haun tehostamiseksi.

Tiettyjen etukäteen tarpeellisiksi tiedettyjen alidokumenttien siirtäminen päätasolle ei kuitenkaan ratkaise ongelmaa kokonaan. Kaikkia tiedontarpeita ei tunneta etukäteen ja koska laajan haun graafisen käyttöliittymän tarkoituksena on korvata vapaa SQL-haku tietokannasta, tarvitaan hakuun toiminnallisuus, jolla hakuehdon voi kohdistaa tietyllä ehdolla esisuodatettuun rakenteen alidokumenttiin.

5. Elasticsearch-haun laajentaminen Taitorakennerekisterissä

Tässä luvussa kerrotaan, minkälaisia tarpeita Taitorakennerekisterissä on laajan haun ilmaisuvoiman lisäämiseen ja kuvaillaan tähän tarpeeseen kehitetyn ratkaisun periaatetta ja sen teknistä toteutusta.

5.1. Haun laajentamistarpeet

Kuten luvuissa kolme ja neljä on kuvailtu, Elasticsearchin sovellusaluekyselykieli mahdollistaa dokumentin alidokumenteista haun vain siten, että dokumentti tulee mukaan tulokseen, mikäli ainakin yksi dokumentin alidokumenteista vastaa annettua ehtoa. Tämä ei kuitenkaan riitä kaikkiin hakutarpeisiin Taitorakenneristerin tapauksessa. Luvussa neljä annettiin esimerkkejä hauista, joiden kaltaisia ei tutkimuksen alkutilanteessa pystynyt Taitorakennerekisterin haulla tekemään, mutta joille käyttäjillä olisi tarvetta. Tässä kohdassa esitellään tarkemmin, millaisia hakuja järjestelmässä pitäisi pystyä muodostamaan.

> Hae rakenteet, joiden pisimmän jänteen väylätyyppi on katu.
> Hae rakenteet, joiden viimeisin sukellustarkastus on tehty ennen vuotta 2000.
> Hae rakenteet, joiden laajin vaurio on kansilaatassa.
> Hae rakenteet, joilla on tasan kaksi halkeilu-tyyppistä vauriota.

Esimerkki 12. Esimerkkejä Taitorakennerekisteriin tarvittavista hauista.

Esimerkissä 12 on vaatimuksia uudelle hakutoiminnallisuudelle. Näistä voi erottaa kaksi erilaista tarvetta, joita tässä tutkimuksessa kutsutaan listan suhteelliseksi suodatuslausekkeeksi ja listan lukumäärälausekkeeksi. Listan suhteellisessa suodatuslausekkeessa valitaan ensin rakenteen tietystä listatyyppisestä arvosta tietyllä kriteerillä alidokumentti (alidokumentin tietty arvo täyttää ehdon suhteessa muihin listan alidokumentteihin) ja testataan, toteuttaako alidokumentti muutoin annetun ehdon tai ehdot. Listan lukumäärälausekkeessa lasketaan listatyyppisen kentän tietyn ehdon toteuttavien alidokumenttien määrä ja testataan, toteuttaako se määrälle asetetun ehdon.

Kyselyitä tulee pystyä vapaasti muodostamaan käyttäen eri alidokumenttityyppejä, arvoja ja ehtoja. Haun ilmaisuvoimaa on laajennettava, koska ei riitä, että pystyttäisiin ainoastaan tekemään esimerkissä 8 esitellyt kyselyt, vaan pitää mahdollistaa myös samankaltaiset mielivaltaisesti määritellyt kyselyt.

5.2. Vertailua kyselyihin SQL-kielellä

Elasticsearchin sovellusaluekyselykielen ilmaisuvoima ei siis perustoiminnallisuudeltaan riitä yllä kuvailtuihin listan suhteelliseen suodatuslausekkeeseen ja listan lukumäärälausekkeeseen. Relaatiotietokannoissa SQL-kielellä ne kuitenkin ovat yksinkertaisia. Tässä esitetään vertailun vuoksi kaksi esimerkin 12 hauista SQL-kielellä.

```
WITH aggregointi AS (
    SELECT rakenneid, MAX(janne.pituus) AS koottutieto
    FROM janne
    GROUP BY rakenneid
)
SELECT DISTINCT janne.rakenneid
FROM
  janne INNER JOIN aggregointi
    ON (janne.rakenneid = aggregointi.rakenneid AND
        janne.pituus = aggregointi.koottutieto)
       WHERE janne.vaylatyyppi_tunnus = 14;
```

Esimerkki 13. SQL-kysely esimerkin 8 haulle ”Hae rakenteet, joiden pisimmän jänteen väylätyyppi on katu”.

```
SELECT rakenne.id
FROM rakenne
  INNER JOIN vaurio
    ON (vaurio.rakenneid = rakenne.id)
WHERE vaurio.tyyppitunnus = 12
GROUP BY rakenne.id
HAVING COUNT(vaurio.tyyppitunnus) = 2;
```

Esimerkki 14. SQL-kysely esimerkin 8 haulle “Hae rakenteet, joilla on tasan 2 halkeilu-tyyppistä vauriota”.

Esimerkissä 13 on esimerkin 12 haku ”Hae rakenteet, joiden pisimmän jänteen väylätyyppi on katu” SQL-kielellä muotoiltuna. SQL-kielellä listan suhteellista suodatuslauseketta vastaava kysely voidaan tehdä esimerkin mukaisesti niin, että muodostetaan väliaikainen taulu, joka on esimerkissä nimetty aggregoinniksi. Sen avulla tunnistetaan kohteet, joille kyselyssä tehdään ehdon tai ehtojen tarkistus.

Esimerkissä 14 on esimerkin 12 hakua ”Hae rakenteet, joilla on tasan 2 halkeilutyyppistä vauriota” vastaava SQL-kysely. SQL:llä listan lukumäärälauseketta vastaavat kyselyt ovat yksinkertaisia, koska aggregointifunktioita voi käyttää kyselyn HAVING-osuudessa.

Listan suhteellisesta suodatuslausekkeesta ja listan lukumäärälausekkeesta voidaan luoda myös geneeriset SQL-kyselyt, joihin taulut, sarakkeet, ehdot ja arvot täydennetään. Jos haku olisi toteutettu SQL-kielellä, tarvittaisiin tällaisissa kyselyissä dynaamista SQL:ää, eli SQL-kyselyn generoimista ajon aikana. Esimerkissä 15 on annettu listan suhteellista suodatuslauseketta vastaava geneerinen SQL-kysely. Dynaamisesti korvattavat osuudet on merkitty kaksoispisteellä alkavilla pienillä aakkosilla kirjoitetuilla sanoilla. Käsitteen tunniste on haettavan entiteetin yksilöivä tunniste tietokannassa, tässä tutkimuksessa tarkastellussa tapauksessa rakenteen id. Aggregointifunktio on alikäsitteeseen (eli tauluun, josta on yhden suhde moneen -tyyppinen viittaus päätauluun) kohdistuva ehto, joka voi olla käytännössä MIN tai MAX. Aggregointiattribuutti määrää alikäsitteen sarakkeen, jonka perusteella alikäsitteen valinta tehdään. Suodatusattribuutti on sarake, johon sovelletaan suodatusehtoa aggregoinnin avulla valitulla alikäsitteen rivillä.

```
WITH aggregointi AS (
    SELECT
      :käsitteentunniste
      :aggregointifunktio(:alikäsite.:aggregointiattribuutti) AS koottutieto
    FROM :alikäsite
    GROUP BY :käsitteentunniste
)
SELECT DISTINCT :alikäsite.:käsitteentunniste
FROM
  :alikäsite INNER JOIN aggregointi ON
     (:alikäsite.:aggregointiattribuutti = aggregointi.koottutieto AND
      :alikäsite.:käsitteentunniste = aggregointi.:käsitteentunniste)
WHERE :alikäsite.:suodatusattribuutti :suodatusehto;
```

Esimerkki 15. Listan suhteellista suodatuslauseketta vastaava geneerinen SQL-kysely.

Esimerkissä 16 on listan lukumäärälauseketta vastaava geneerinen SQL-kysely. Käsite on päätaulu, josta haetaan, tässä esimerkissä rakenne. Käsitteen tunniste ja alikäsite ovat merkitykseltään samoja kuin listan suhteellista suodatuslauseketta kuvaavassa kyselyssä. Suodatusattribuutti määrää alikäsitteen sarakkeen, jonka arvoon suodatusehtoa sovelletaan. Aggregointiehdolla ja lukumäärällä valitaan, kuinka monen alikäsitteen rivin on noudatettava suodatusehtoa.

```
SELECT :käsite.:käsitteentunniste
FROM :käsite
INNER JOIN :alikäsite ON (:käsite.:käsitteentunniste = :alikäsite.:käsitteen-
tunniste)
WHERE :alikäsite.:suodatusattribuutti :suodatusehto
GROUP BY :käsite.:käsitteentunniste
HAVING COUNT(:alikäsite.:suodatusattribuutti) :aggregointiehto :lukumäärä;
```

Esimerkki 16. Listan lukumäärälauseketta vastaava geneerinen SQL-kysely.

5.3. Ratkaisun tekninen toteutus

Vaikka Elasticsearchin sovellusaluekyselykielen perustoiminnallisuus ei mahdollistakaan monipuolista hakua alidokumenteista, yllä kuvattujen kyselyjen toteutus on mahdollista Elasticsearchin skriptausominaisuuden avulla. Skriptauksella voidaan Elasticsearch-kyselyssä suodattaa hakutuloksia, vaikuttaa niiden pisteytykseen ja muokata palautuvan hakutuloksen arvoja [Elastic, 2017b]. Kyselyssä käytetyt skriptit voivat olla joko kirjoitettuna kyselyyn (inline), indeksoituna skripteille tarkoitettuun sisäiseen indeksiin tai ne voidaan sijoittaa tiedostoihin, jolloin skripteihin viitataan kyselyssä niiden nimillä.

Näistä tavoista tietoturvan kannalta suositeltavin on skriptien sijoittaminen tiedostoihin. Mikäli sallitaan skriptien kirjoittaminen suoraan kyselyyn, kyselyn suorittavalle käyttäjälle annetaan käytännössä oikeudet suorittaa mitä tahansa ohjelmakoodia Elasticsearchin käytössä olevalla tietokoneella, mikä on tietoturvan kannalta ongelmallista. Elasticsearchissa voidaan käyttää myös *hiekkalaatikkoon rajoitettuja* (sandboxing) kieliä, jotka ovat turvallisia käyttää suoraan kyselyyn kirjoitetusta skriptistäkin. Elasticsearchin versiosta 5.0 alkaen ohjelmistossa mukana tuleva Painless-kieli on tällainen. Taitorakennerekisterissä käytössä olevassa versiossa vaihtoehdot ovat kuitenkin rajatumpia, eikä tarkoitukseen sopivaa hiekkalaatikkoon rajoitettua kieltä ole käytettävissä.

Taitorakennerekisterin hakutarpeita varten toteutetussa ratkaisussa kyselyjä täydennetään hakutulosta suodattavalla Groovy-ohjelmointikielellä kirjoitetulla skriptillä. Groovy on Javaan perustuva, monipuolinen ja skriptaukseen erityisesti suunniteltu ohjelmointikieli, jota Elasticsearch oletuksena käyttää skriptauskielenä. Hakutuloksesta siis suodatetaan skriptillä pois dokumentit, joita ei haluta mukaan hakutulokseen. Suodatuksessa käytettävä skripti käsittelee yhden hakutuloksen. Skriptissä voidaan lukea tietoja käsiteltävästä hakutuloksesta ja skripti palauttaa totuusarvon sen mukaan, hyväksytäänkö dokumentti hakutulokseen vai ei. Skriptille voi myös antaa parametreja osana kyselyä.

Listan suhteellista suodatuslauseketta varten toteutettu skripti saa parametreina alidokumentteja sisältävän kentän, josta haetaan, suhteelliseen suodatukseen käytettävän kentän, aggregointifunktion ja kentän, suodatusoperaattorin sekä verrattavan arvon, jolla testataan ehdon toteutuminen suodatetussa alidokumentissa. Skriptissä käydään läpi parametrilla määritellyt alidokumentit ja suodatetaan niistä se, joka täyttää aggregointiehdon (esimerkiksi pisin jänne tai viimeisin tarkastus) ja lopuksi tarkistetaan, täyttääkö suodatettu alidokumentti annetun toisen ehdon. Jos täyttää, palautetaan tosi, jolloin dokumentti hyväksytään hakutulokseen. Muutoin palautetaan epätosi. Esimerkissä 17 on esitetty listan suhteellisen suodatuslausekkeen skriptin toiminta pseudokoodilla.

```
ListanSuhteellinenSuodatusLauseke(alidokumentti, suodatuskenttä, aggregointifunktio
(min/max), aggregointikenttä, suodatusoperaattori, vaadittu-arvo)

alidokumentit=dokumentti.alidokumentti;
alidokumentti = first(alidokumentit);
for(verrattava_alidokumentti in alidokumentit)
      alidokumentti = aggregointifunktio(alidokumentti, verrattava_alidokumentti)
if(alidokumentti.suodatuskenttä suodatusoperaattori vaadittu-arvo)
      return true;
else
      return false;
```

Esimerkki 17. Listan suhteellisen suodatuslausekkeen hakutulosten suodattamiseen tarvittava skripti pseudokoodina.

```
{"query": {"bool": {"must":
                     [{"nested":
                        {"path": "janne",
                           "query": {
                             "bool":
                               "must": [
                                {"bool": {
                                   "must": {
                                     "exists":{"field":"janne.pituus"}}}},
                                {"bool":{"must":{"terms{"janne.vaylatyyppi:
                                                          ["14"]}}}}]}}}},
                                {"filtered": {
                                   "filter": {
                                     "script": {
                                       "script": {
                                         "file": "listan_suhteellinen_suodatus_lauseke",
                                                 "params": {
                                                   "alidokumentti": "janne",
                                                   "suodatuskenttä": "vaylatyyppi",
                                                   "aggregointifunktio": "max",
                                                   "aggregointikenttä": "pituus",
                                                   "suodatusoperaattori": "joukossa",
                                                    "vaadittu-arvo":
                                                    ["14"]}}}}]}}}]}}}}
```

Esimerkki 18. Esimerkki listan suhteellisesta suodatuslausekkeesta Elasticsearchin sovellusaluekyselykielellä.

Skriptin kutsumisen lisäksi kyselyssä rajataan hakutuloksesta ne dokumentit, joissa skriptin tarkastamat ehdot eivät voi toteutua. Tämä tehostaa kyselyä, kun ei tarvitse käydä skriptissä läpi kaikkia dokumentteja. Esimerkissä 18 on listan suhteellinen suodatuslauseke Elasticsearchin sovellusaluekyselykielellä. Haussa etsitään rakenteet, joiden pisimmän jänteen väylätyyppi on katu (tunnus 14). Kyselyssä haetaan ensin rakenteet, joilla on ainakin yksi jänne, jolla on pituus ja ainakin yksi jänne, jonka väylätyyppi on katu. Sen jälkeen hakutulokset suodatetaan skriptin avulla ja saadaan tulokseksi rakenteet, joiden pisimmän jänteen väylätyyppi on katu.

Listan lukumäärälausekkeen toteuttavan skriptin toimintaperiaate on samanlainen. Se saa parametreina alidokumentteja sisältävän kentän, josta haetaan, tarkistukseen käytettävän alidokumentin kentän, operaattorin ja verrattavan arvon, sekä aggregointifunktion tuottaman arvon vertailuun käytettävän operaattorin ja verrattavan lukumäärän. Skriptissä käydään läpi kaikki annetun kentän alidokumentit ja lasketaan, kuinka moni näistä alidokumenteista täyttää annetun ehdon. Lopuksi tarkistetaan, täyttikö saatu lukumäärä lukumäärälle asetetun ehdon ja palautetaan totuusarvo sen mukaan. Esimerkissä 19 on listan lukumäärälausekkeen suodatukseen käytettävä skripti pseudokoodina. Todellisuudessa skripteihin tuo monimutkaisuutta se, että operaattorit on välitettävä skriptille merkkijonoina, jolloin jokaista operaattoria varten pitää olla oma haara koodissa.

```
ListanLukumääräLauseke(alidokumentti, kenttä, operaattori, verrattava-arvo,
aggregointioperaattori, verrattava-lukumäärä)

laskuri=0;
for(alidokumentti in dokumentti.alidokumentti)
     if(alidokumentti.kenttä operaattori verrattava-arvo)
          laskuri++;
if(laskuri aggregointioperaattori verrattava-lukumäärä)
     return true;
else
     return false;
```

Esimerkki 19. Listan lukumäärälausekkeen suorittamiseen tarvittava skripti pseudokoodina.

Samoin kuin listan suhteellisessa suodatuslausekkeessa, listan lukumäärälausekkeessakin ensin karsitaan pois dokumentit, jotka eivät voi täyttää kyselyn skriptiosassa tarkistettavia ehtoja. Näin läpi käytävien dokumenttien määrä vähenee. Esimerkissä 20 on listan lukumäärälauseke Elasticsearchin sovellusaluekyselykielellä. Kyselyssä haetaan ensin rakenteet, joilla on ainakin yksi tarkastus, jonka kommentit ja puutteet –kentässä esiintyy jossakin kohtaa merkkijono ”puuttuu”. Tämän jälkeen valitaan hakutulokseen skriptin avulla vain rakenteet, joilla on vähintään kaksi tällaista tarkastusta.

```
{"query":
 {"bool":
   {"must": [{
      "bool": {
       "should": [{
         "bool": {
           "must": [{
             "nested": {
               "path": "tarkastus",
                "query": {
                  "bool": {
                    "must": [{
                      "bool": {
                        "must": {
                          "wildcard":{"tarkastus.kommentitjapuutteet":
                                        "*puuttuu*"}}}},
                {"filtered":
                  {"filter":
                    {"script": {
                       "script": {
                       "file": "listan_lukumaara_lauseke",
                       "params": {"alidokumentti": "tarkastus",
                                  "kentta": "kommentitjapuutteet",
                                  "operaattori": "kuin",
                                  "verrattava-arvo": "puuttuu",
                                  "aggregointioperaattori": ">",
                                  "verrattava-lukumaara": 2}}}}]}}}]}}}]}}}}}
```

Esimerkki 20. Listan lukumäärälauseke Elasticsearchin sovellusaluekyselykielellä.

Taitorakennerekisterissä kyselyllä on Elasticsearchin sovellusaluekyselykielestä poikkeava sisäinen esitysmuoto. Sisäinen kysely muodostetaan laajan haun käyttöliittymän (kuva 2 luvussa 4) valintojen perusteella. Sisäinen esitysmuoto puolestaan muunnetaan sovellusaluekyselykielen kyselyksi ennen suorittamista. Luvun neljä esimerkissä 1 on kysely järjestelmän sisäisessä esitysmuodossa. Käyttöliittymässä rivit on erotettu JA-operaattorilla ja sarakkeet TAI-operaattorilla. Sama logiikka on myös järjestelmän sisäisessä kyselyn esitysmuodossa. Elasticsearch-kyselyssä tämä toteutetaan **bool** –rakenteella ympäröimällä rivin sisältämät sarakkeet **should** –rakenteella ja sarakkeen sisältämät lauseet **must** –rakenteella.

Lausekkeen kohdalla muuntaminen riippuu lausekkeen tyypistä. Yksiarvoisessa lausekkeessa on kenttä, operaattori ja verrattava arvo, ja se muunnetaan Elasticsearchin sovellusaluekyselykielen vastaavaksi kyselyksi. Listan jokin –tyyppisessä kyselyssä haetaan listatyyppisestä kentästä niin, että ainakin yksi listan alidokumenteista täyttää kyselyn ehdot. Tällainen kysely muunnetaan Elasticsearchin sovellusaluekyselykielen **nested** –kyselyksi. Luvun neljä esimerkissä 7 on **nested** –kysely.

Listan suhteellisen suodatuslausekkeen ja listan lukumäärälausekkeen muuntamisessa itse kysely muunnetaan samaan tapaan kuin listan jokin –lausekkeen kohdalla. Tällä tavoin muodostetussa kyselyssä hakutulokseen ensin valitaan ne dokumentit, joilla on

ainakin yksi ehdot täyttävä alidokumentti. Listan jokin –tyyppisen alikyselyn lisäksi kyselyyn lisätään **filtered**-rakenne, jonka sisällä kutsutaan suodattamisen suorittavaa skriptiä. Kuten yksiarvoinen ja listan jokin –lauseke, myös listan suhteellinen suodatuslauseke ja listan lukumäärälauseke voivat olla osana laajempaa kyselyä **must** tai **should**-rakenteen sisällä.

6. Toteutetun ratkaisun arviointia

Tässä luvussa arvioidaan Taitorakennerekisterin hakuun toteutetun ilmaisuvoimaa lisäävän ratkaisun onnistumista ja sen puutteita ja pohditaan, miten ratkaisua voisi kehittää edelleen. Luvussa pohditaan myös, mitä muita tapoja tässä tutkimuksessa kuvatun ongelman ratkaisemiseen voisi käyttää.

Kuten alakohdassa 2.2.1 kuvailtiin Marchin ja Smithin [1995] ja Järvisen ja Järvisen [2001] pohjalta, olennainen osa konstruktiivista tutkimusta on toteutetun innovaation arviointi. Toteutettua innovaatiota on aina verrattava vastaaviin aiempiin innovaatioihin. Tässä tutkimuksessa toteutettua ratkaisua vastaa parhaiten Pasqualinin ja muiden [2016] ratkaisu, joka käsittelee alidokumentteihin kohdistuvia hakuja ja aggregaatiota NoSQL-tietokannoissa. Pasqualinin ja muiden ratkaisussa alidokumenttien aggregointitietoja indeksoitiin suoraan dokumentin tasolle. Tässä tutkimuksessa toteutettu ratkaisu mahdollistaa ennalta tuntemattomiin hakutarpeisiin vastaamisen, mitä Pasqualinin ja muiden lähestymistapa ei mahdollista. Toisaalta, jos hakutarpeet tunnetaan hyvin etukäteen, Pasqualinin ja muiden lähestymistapa on todennäköisesti yksinkertaisempi toteuttaa ja tehokkaampi, koska haettaessa ei enää tarvitse käydä alidokumentteja läpi ohjelmallisesti.

Tässä tutkimuksessa konstruktiivisen tutkimuksen tuloksena oli realisaatio. Marchin ja Smithin [1995] mukaan realisaatiota arvioidaan sen tehokkuuden ja vaikuttavuuden kautta, sekä arvioidaan sen vaikutuksia ympäristöön ja käyttäjiin. Järvinen ja Järvinen [2001] lisäävät näihin arviointikriteereihin vielä laajemmat vaikutukset, joita artefaktilla ja sen käyttöönotolla on sekä elinkaareen liittyvät asiat, kuten huollot tarpeen ja käytöstä poiston vaikutukset.

Tutkimuksessa toteutetun toiminnallisuuden tehokkuus on kontekstissaan riittävän hyvä. Tehokkuutta laskee se, että hakutulosta suodattaessa skripteillä on dokumentin alidokumentit aina käytävä läpi ohjelmallisesti. Tämän vuoksi Elasticsearch ei myöskään voi optimoida suoritustaan skriptejä käytettäessä yhtä tehokkaasti kuin tavanomaisten kyselyiden kohdalla. Elastic [2017b] korostaakin sovelluskehittäjiä ottamaan huomioon skriptitoiminnallisuutta käytettäessä tehokkuuden. Taitorakennerekisterin hakuindeksissä alidokumentteja on kuitenkin korkeintaan satoja. Isoa määrää dokumentteja ei myöskään yleensä suodatusvaiheessa käydä läpi, koska kyselyllä on jo ennen suodatusta karsittu pois suuri määrä dokumenteista.

Toteutetun ratkaisun vaikuttavuutta on tässä vaiheessa vaikea arvioida, koska se ei ole vielä käytössä Taitorakennerekisterissä. Haun tekemiseksi järjestelmässä tulisi olla käyttöliittymä, jolla kysely muotoiltaisiin. Käyttöliittymä rajattiin kuitenkin tämän tutkimuksen ulkopuolelle, eikä sitä ole vielä toteutettu. Käytettävyydeltään hyvän käyttöliittymän toteuttaminen tämän tutkimuksen aiheena oleviin käyttötapauksiin tulee todennäköisesti olemaan haastavaa.

Toteutettu toiminnallisuus kuitenkin kattaa suuren määrän ominaisuuksista, joita hakutoiminnallisuuden laajennukselta alun perin vaadittiin. Kaikki esimerkissä 12 annetut haut voidaan tehdä toteutetun toiminnallisuuden avulla. Nämä haut voidaan myös yhdistää osaksi laajempaa kyselyä Boolen operaattoreita hyödyntäen. Toteutetun ratkaisun kattavuudessa on kuitenkin myös puutteita. Koska ratkaisussa käytetään skriptejä, mahdolliset uudet hakutapaukset ovat tiukasti tietyn kaavan mukaisia. Toteutetulla ratkaisulla ei tällä hetkellä ole esimerkiksi mahdollista antaa useampia kuin yksi ehto suodatetulle alidokumentille kummassakaan uudessa kyselytyypissä. Tämän mahdollistamiseksi suodattamiseen käytettävää skriptiä tulisi laajentaa. Listan lukumäärälausekkeessa (luvun viisi esimerkit 19 ja 20) myös oletetaan, että haettava lukumäärä on positiivinen. Jos haluttaisiin löytää esimerkiksi sillat, joilla ei ole yhtään vauriota kansilaatassa, pitäisi tällaiselle kyselylle myös tehdä erillinen skripti.

Vaikutusta ympäristöön ja käyttäjiin ei myöskään ole kovin helppo arvioida tässä vaiheessa. Kun toteutus on otettu käyttöön, se todennäköisesti vastaa joidenkin järjestelmän käyttäjien tiedontarpeisiin ja helpottaa näin tarvittavan tiedon löytämistä järjestelmästä. Elastic [2017b] kehottaa huomioimaan skriptejä käytettäessä myös tietoturvan. Toteutetussa ratkaisussa skriptien käyttö on kuitenkin turvallista, koska ajettavat skriptit tallennetaan tiedostoihin ja niiden toimintaan vaikutetaan vain parametrien kautta. Käyttäjä ei siis pääse ajamaan mielivaltaista koodia skriptien muodossa.

Edellä mainittu toteutetun ratkaisun jäykkyys aiheuttaa myös sen, että sitä on kehitettävä käyttäjien tiedontarpeiden mukaan. Ratkaisua on vietävä iteratiivisesti oikeaan suuntaan käyttäjien palautteen perusteella. Toteutettua toiminallisuutta voisi käyttöliittymän suunnittelun ja toteuttamisen jälkeen kehittää laajentamalla skriptien toimintaa esimerkiksi niin, että suodatetulle alidokumentille voisi antaa useampia ehtoja. Myös uusia haun laajentamisen tarpeita voisi kartoittaa ja tehdä uusia skriptejä ja kyselytyyppejä sen pohjalta.

Tässä tutkimuksessa ratkaistun ongelman voisi ratkaista myös muilla tavoilla kuin luvussa 5 esitetyllä ratkaisulla. Taitorakennerekisterin haussa on jo ennestään käytetty samankaltaista menetelmää kuin Pasqualin ja muiden [2016] ratkaisussa, eli joidenkin alidokumenttien kenttien indeksoimista päätasolle ennalta tiedettyjen hakutarpeiden perusteella. Esimerkiksi rakenteen viimeisimmän tarkastuksen tietoja on indeksoitu omana kenttänään rakenteessa. Tätä toiminnallisuutta olisi mahdollista myös laajentaa niin, että uusia kenttiä indeksoitavaksi päätasolle ja niihin liittyviä alidokumenteille asetettuja ehtoja voisi luoda käyttöliittymässä. Tämä vaatisi puolestaan SQL:n generoimista, jotta tarvittavat tiedot saataisiin vietyä tietokannasta mukaan indeksointiin.

Toteutetun ratkaisun voidaan siis arvioida täyttäneen sille asetetut vaatimukset ja olevan riittävän tehokas ja tietoturvallinen. Toteutuksen vaikuttavuutta ja sen vaikutuksia

käyttäjiin on vielä mahdoton arvioida kattavasti, kun toteutusta ei ole vielä otettu käyttöön järjestelmässä. Toteutus vaatii vielä paljon kehittämistä käyttäjien palautteen perusteella ja soveltamista erilaisiin hakuihin.

7. Yhteenveto

Tässä tutkimuksessa toteutettiin Taitorakennerekisteri-järjestelmän laajan haun toiminnallisuuteen mahdollisuus tehdä dokumenttien listatyyppisiä alidokumentteja suodattavia hakuja sekä alidokumenttien aggregointiin perustuvia hakuja. Tämän tavoitteen saavuttamiseksi myös tutkittiin, millaisia ratkaisuja kirjallisuudessa on kuvattu alidokumenttihakuihin liittyen NoSQL-tietokannoissa. Toteutetussa ratkaisussa hyödynnettiin Elasticsearchin skriptaustoiminnallisuutta, koska sen avulla saatiin toiminnallisuudelle asetetut vaatimukset täytettyä mahdollisimman kattavasti.

Tutkimuksen tuloksista voi päätellä, että vaikka NoSQL-tietokannat ja hakukoneet ovat relaatiotietokantoja tehokkaampia joissakin käyttötapauksissa ja yksinkertaisten hakujen tekeminen on niillä suoraviivaista, monimutkaisempien hakujen kohdalla joudutaan niiden puutteita joskus korvaamaan sovelluslogiikalla ja ylimääräisellä ohjelmakoodilla. Elasticsearchin tapauksessa kuitenkin monimutkaisempien hakujenkin toteuttaminen on mahdollista ja skriptien avulla toiminnallisuutta voi muokata hyvin monipuolisesti halutunlaiseksi.

Tutkimuksessa toteutettua ratkaisua tulee kehittää edelleen ja sen yhteyteen tulisi toteuttaa käytettävyydeltään hyvä käyttöliittymä. Tällöin myös itse toiminnallisuutta voitaisiin kehittää eteenpäin käyttäjäpalautteen perusteella.

Olisi kiinnostavaa tutkia, onko vastaavaa toiminnallisuutta mahdollista toteuttaa muissa dokumenttivarastoissa ja myös muun tyyppisissä NoSQL-tietokannoissa. Samoin voisi tutkia, onko vastaavia hakuja mahdollista tehdä muissa hakukoneissa. Mielenkiintoista olisi myös kartoittaa, millaisia erilaisia monimutkaisempia hakuja Elasticsearchilla on mahdollista tehdä ja laajentaa käsittelyä tässä tutkimuksessa esiteltyjen käyttötapausten ulkopuolelle. Voisi olla hyödyllistä myös tutkia erilaisten sovellusalueiden vastaavia käyttötapauksia.

Viiteluettelo

[Amazon Web Services, 2017] Amazon Web Services: Amazon Simple DB. https://aws.amazon.com/simpledb/ Checked 4.5.2017.

[Apache Software Foundation, 2017a] Apache Software Foundation: Apache CouchDB. http://couchdb.apache.org/ Checked 4.5.2017.

[Apache Software Foundation, 2017b] Apache Software Foundation: Apache HBase. http://hbase.apache.org/ Checked 4.5.2017.

[Apache Software Foundation, 2017c] Apache Software Foundation: Apache Cassandra. http://cassandra.apache.org/ Checked 4.5.2017.

[Atzeni *et al*., 2012] Paolo Atzeni, Francesca Bugiotti and Luca Rossi, SOS (save our systems): a uniform programming interface for non-relational systems. In: *Proc. of the 15th International Conference on Extending Database Technology* (Mar 2012), 582-585.

[Atzeni *et al.*, 2013] Paolo Atzeni, Christian S. Jensen, Giorgio Orsi, Sudha Ram, Letizia Tanca and Riccardo Torlone, The relational model is dead, SQL is dead and I don't feel so good myself. *ACM SIGMOD Record* **42**, 2 (Jun 2013), 64-68.

[Basho, 2017] Basho: Riak KV. http://basho.com/products/riak-kv/ Checked 4.5.2017.

[Baxter and Jack, 2008] Pamela Baxter and Susan Jack, Qualitative case study methodology: study design and implementation for novice researchers. *The Qualitative Report* **13,** 4 (Jan 2008), 544-449.

[Bray, 2014] Bray, T. RFC 7159-The JavaScript object notation (JSON data interchange format, 2014.

[Cattell, 2010] Rick Cattell, Scalable SQL and NoSQL data stores. *ACM SIGMOD Record* **15**, 4 (Dec 2010), 12-27.

[Chang *et al.*, 2008] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber, Bigtable: a distributed storage system for structured data. *ACM Transactions on Computer Systems* **26**, 4 (Jun 2008), 1–26.

[Chattopadhyay *et al.*, 2011] Biswapesh Chattopadhyay, Prathyusha Aragonda, Liang Lin, Vera Lychagina, Weiran Liu, Younghee Kwon, Sagar Mittal and Michael Wong, Tenzing – A SQL implementation on the MapReduce framework. In: *Proc. of the VLDB Endowment* (Sep 2011), 1318-1327.

[Clojure.org, 2017] Rich Hickey, http://clojure.org. Checked 10.1.2017.

[ClojureScript.org, 2017] http://clojurescript.org. Checked 11.1.2017.

[Codd, 1970] Edgar T. Codd, A relational model of data for large shared databanks. *Communications of the ACM* **13**, 6 (Jun 1970), 377-387.

[Dean and Ghemawat, 2008] Jeffrey Dean and Sanjay Ghemawat, MapReduce: simplified data processing on large clusters. *Communications of the ACM* **51**, 1 (Jan 2008), 107-113.

[DeCandia *et al.*, 2007] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels, Dynamo: Amazon's highly available key-value storage. In: *Proc. of twenty-first ACM SIGOPS* (Oct 2007), 205-220.

[Elastic, 2017a] Elasticsearch Reference: Bool Query. https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html Checked 16.1.2017.

[Elastic, 2017b] Elasticsearch Reference: Scripting. https://www.elastic.co/guide/en/elasticsearch/reference/2.3/modules-scripting.html Checked 24.1.2017.

[Elastic, 2017c] Elasticsearch Glossary. https://www.elastic.co/guide/en/elasticsearch/reference/current/glossary.html Checked 3.2.2017.

[Elastic, 2016] Elasticsearch: The Definitive Guide. https://www.elastic.co/guide/en/elasticsearch/guide/master/dynamic-mapping.html Checked 30.12.2016.

[Gao *et al.*, 2011] Xiaoming Gao, Vaibhav Nachankar and Judy Qiu, Experimenting Lucene index on HBase in an HPC environment. In: *Proc. of the first annual workshop on high performance computing meets databases* (Nov 2011), 25-28.

[Haerder and Reuter, 1987] Theo Haerder and Andreas Reuter, Principles of transaction-oriented database recovery. *ACM Computing Surveys* **15**, 4 (Dec 1987), 287-317.

[Hecht and Jablonski, 2011] Robin Hecht and Stefan Jablonski, NoSQL evaluation: A use case oriented survey. In: *Proc. of the 2011 International Conference on Cloud and Service Computing* (Dec 2011), 336-341.

[Hypertable Inc, 2017] Hypertable. http://www.hypertable.org/ Checked 4.5.2017.

[Järvinen ja Järvinen, 2001] Pertti Järvinen ja Annikki Järvinen, *Tutkimustyön metodeista*. Opinpajan kirja, 2001.

[Kononenko *et al.*, 2014] Oleksii Kononenko, Olga Baysal, Reid Holmes and Michael W. Godfrey, Mining modern repositories with Elasticsearch. In: *Proc. of the 11th Working Conference on Mining Software Repositories* (May – Jun 2014), 328-331.

[Kroenke and Auer, 2016] David M. Kroenke and David J. Auer, *Database Processing: Fundamentals, Design and Implementation.* Pearson, 2016.

[Liikennevirasto, 2013] Minna Torkkeli, Marja-Kaarina Söderqvist, Risto Lång, Mauno Alaluusua, Tero Sikiö, Veli-Matti Hirvonen, Ilkka Kuulas, Matti Piispanen, Pekka

Siitonen, Timo Tirkkonen, Olli-Pekka Aalto ja Ari Savolainen, Taitorakenteiden tarkastusohje. Liikenneviraston ohjeet, huhtikuu 2013.

[March and Smith, 1995] Salvatore March and Gerald Smith, Design and natural science research on information technology, *Decision Support Systems* **15** (1995), 251-266.

[Memcached, 2017] Memcached. https://memcached.org/ Checked 4.5.2017.

[Mohan, 2013] C. Mohan, History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla. In: *EDBT 2013* (Mar 2013), 11-16.

[MongoDB Inc, 2017] MongoDB. https://www.mongodb.com/ Checked 4.5.2017.

[Neo Technology Inc, 2017] Neo Technology Inc: Neo4j. https://neo4j.com/ Checked 4.5.2017.

[Ontotext, 2017] Ontotext: GraphDB. http://ontotext.com/products/graphdb/ Checked 4.5.2017.

[OrientDB Ltd, 2017] OrientDB. http://orientdb.com/ Checked 4.5.2017.

[Parker *et al.*, 2013] Zachary Parker, Scott Poe and Susan V. Vrbsky, Comparing NoSQL MongoDB to an SQL DB. In: *Proc. of the 51st ACM Southeast Conference* (Apr 2013), 5:1--5:6.

[Pasqualin *et al.*, 2016] Diago Pasqualin, Giovanni Souza, Eduardo Luis Buratti, Eduardo Cunha de Almeida, Marcos Didonet Del Fabro and Daniel Weingaertner, A case study of the aggregation query model in read-mostly NoSQL document stores. In: *Proc. of the 20th International Database Engineering & Applications Symposium* (Jul 2016), 224-229.

[Pokorný, 2011] Jaroslav Pokorný, Database technologies in the world of big data. In: *Proc. of the 16th International Conference on Computer Systems and Technologies* (Jun 2015), 1-12.

[Pritchett, 2008] Dan Pritchett, BASE: An acid alternative. *Queue – Object Relational Mapping* **6**, 3 (May/Jun 2008), 48-55.

[Project Voldemort, 2017] Project Voldemort. http://www.project-voldemort.com/voldemort/ Checked 4.5.2017.

[Redis, 2017] Redis. https://redis.io/ Checked 4.5.2017.

[Rith *et al.*, 2014] Julian Rith, Philipp S. Lehmayr and Klaus Meyer-Wegener, Speaking in tongues: SQL access to NoSQL systems. In: *SAC'14* (Mar 2014), 855-857.

[Solita Oy, 2014] Teemu Ilkka ja Harri Ohra-aho, Teknologiavalinnat. Lokakuu 2014. (Ei julkaistu.)

[Solita Oy, 2015] Ville Vehviläinen ja Teemu Ilkka, Toiminnalliset kokonaisuudet. Syyskuu 2015. (Ei julkaistu.)

[Solita Oy, 2016] Ville Vehviläinen ja Teemu Ilkka, Haun toteutus. Maaliskuu 2016. (Ei julkaistu.)

[Stake, 1995] Robert E. Stake, *The Art of Case Study Research*. Sage, 1995.

[Stonebraker *et al.*, 2007] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem and Pat Helland, The end of an architectural era (it's time for a complete rewrite). In: *VLDB 2007* (Sep 2007), 1150-1160.

[Tunkelang, 2009] Daniel Tunkelang, *Faceted Search*. Morgan & Claypool, 2009.

[Yin, 2003] Robert K. Yin, *Case Study Research – Design and Methods*. Sage, 2003.

[Zuse Institute Berlin, 2017] Zuse Institute Berlin: Scalaris. http://scalaris.zib.de/ Checked 4.5.2017.